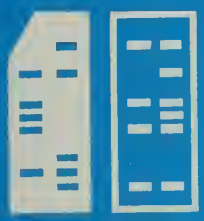
Report No. UIUCDCS-R-74-657 NSF - OCA - GJ-36936 - 000005

AN EXPERIMENTAL INFORMATION RETRIEVAL SYSTEM

by

William Howard Stellhorn

July 1974



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. UIUCDCS-R-74-657

AN EXPERIMENTAL INFORMATION RETRIEVAL SYSTEM*

by

William Howard Stellhorn

July 1974

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the National Science Foundation under Grant No. US NSF-GJ-36936.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

PART I. User's Guide

1.0 Introduction

This report describes the initial implementation of an experimental system designed to support the development of effective algorithms for retrieving information from a large variety of data bases, and especially from data bases which have very little inherent structure. Because direct sequential scanning of the data is expected to play some part in the operation of such a retrieval system, an immediate goal of this program is to evaluate quantitatively various strategies for efficient searching in this environment. Another immediate goal is to study the interaction between the system and a group of motivated users who are not necessarily experts either in computer techniques or in the subject matter of the data base.

In this implementation, searching is performed by means of a sequential scan through the complete text of the data base. The user has very general control over the text of the search terms, the logical structure of the search request and the contexts to be searched and printed. Any character string may be used as a search term, and it is possible to locate co-occurrences of terms within sentences or paragraphs as well as in larger document subdivisions.

The system runs on a microprogrammable Burroughs D-Machine minicomputer with 1024, 64-bit words of microstore and 8192, 16-bit words of main memory. Peripherals include a card reader, a line printer and a disk with a storage capacity of about 25 million bits.

A separately-developed, companion system, which uses an inverted file organization and a considerably expanded inquiry language and which runs

on a DEC PDP-11, is also operational. Eventually the two systems might be linked together.

Part I of this report is a user's manual which describes the organization of data and the use of the inquiry language in the D-Machine retrieval system. Part II describes the operating details of the control program with special attention to the algorithms which schedule and control searching and which may be used to investigate search strategies.

2.0 Data Base Organization

The data base is organized as a collection of "documents", each of which may be divided into several sections and subsections as required by a particular application. The remainder of this discussion will deal with a collection of technical articles although the data formatting system described could be applied equally well to other contexts.

Each article enters the system directly in its full original form with a number of special characters, or context delimiters, inserted for the purpose of identifying the beginnings and ends of the various sections. These context delimiters are arranged in the hierarchical structure illustrated in Figure 2.1. Words printed in capital letters in the figure are section names, and any of these names may be specified by the user as an area to be searched or printed by means of commands to be described in Section 3.0. Two or more names shown together on a single line are synonymous and may be used interchangeably. When one context name in the figure is indented relative to another, the former section is completely contained within the latter, and is implicitly included in any reference to the latter. For example, TEXT includes ABSTRACT, BODY, and NOTES but excludes reference lists, index terms, bibliographic data, etc. The terms DOCUMENT and ARTICLE are interchangeable and include all other sections.

Using this system, a person may restrict his search to titles, author's names, full bibliographic citations, abstracts, keyword lists, etc.; or he may include the entire contents of the data base. Searching can also be performed within sentences or paragraphs, in which case each sentence (paragraph) in Sections D, E and F is searched individually in accordance with the search request.

DOCUMENT or ARTICLE

A. DATA (bibliographic)
   1. AUTHOR
   2. TITLE
   3. SOURCE (publication)
   4. DATE
   5. PAGE or PAGES
   6. MISC (any other bibliographic data in the file)

B. INDEX (not currently used--may be used later for a concordance)

C. KEYS (keyword list published as part of the document)

D. TEXT
   1. ABSTRACT
   2. BODY (text of document)
   3. NOTES (footnotes)

E. REFERENCES or REFS

F. COMMENTS (reserved for user's comments to be recorded with the file)

Items D, E, and F may be further subdivided by PARAGRAPH and SENTENCE.

Figure 2.1. Document Structure

Some terms in Figure 2.1 require explanation. Sections B and C (INDEX, KEYS) are reserved tentatively for two different systems of index terms which might be attached to the document by the author, an independent agency or the retrieval system itself. Neither section is used at the present time, and they are available mainly for experimental purposes and for system growth. Section F (COMMENTS) is provided for use with a note-taking facility which may be added to the system at a later date. This would allow a user to enter comments of his own for future retrieval with a document. Such comments might be stored permanently with the original text (in a small private system), or they might be loaded from a user's personal files during the LOGON procedure.

3.0 Interactive Search Language

The interactive language provides for communication between the user and the retrieval program. Two instructions, FIND and PRINT, are currently available; and several others which require the use of disk facilities will be provided soon. Throughout this description of the search language instructions, the symbols "< >" are used to indicate that some information is to be supplied by the user; and square brackets, [ ], are used to indicate that the information enclosed is optional. These symbols are never part of the required input. The keywords "FIND", "PRINT", and "IN" must always be used as shown in the sample instructions and must be followed by at least one blank. Other blanks are ignored except when they appear between apostrophes as part of a search term.

3.1 FIND Instruction

The FIND instruction causes the system to search for the occurrence of one or more character strings specified by the user. The search may be conducted in the entire document or may be restricted to particular sections or subsections. The format for the FIND instruction is:

FIND < logical expression > [IN < context name >].

The "logical expression" required here consists of character strings to be located in the text, enclosed by apostrophes, and separated by the symbols "*" (logical AND) and "+" (logical OR). Parentheses may be used freely to group terms in any way desired by the user. In the absence of parentheses, the operator "*" is considered to be dominant over the operator "+", i.e.,

X + Y * Z

is equivalent to

X + (Y*Z).

Connecting two or more search strings by "*" will cause a document to be retrieved only if all the strings so connected are present together in the context specified. Joining two or more strings by "+" will cause retrieval of a document if any of the requested strings is present. Currently, the logical operator "NOT" is not available, although provision has been made in the control program to include it at a later time.

Since the retrieval program was designed to operate on many different kinds of data bases, there is no restriction on the character strings which may be sought except that the requested string must be entered exactly as it appears in the text. For example, the string 'SHAR' could be used to retrieve any or all of the following: SHARE, SHARED, SHARING, TIMESHARING, etc. These strings need not observe word boundaries; prefixes, suffixes, or both may be dropped; punctuation marks may be included; and the strings requested may overlap. Note, however, that a search string may not extend from one sentence or paragraph to another because these context units are separated in the text by special characters which cannot be entered from the input terminal. Since apostrophes are used to indicate the ends of the search strings, apostrophes which are to be included in the search itself must be typed twice. For example, to locate the word ISN'T, type: FIND 'ISN''T' ... .

The use of IN followed by a context name is an optional feature which allows the user to restrict his search to selected document sections as defined in Figure 2.1. By means of the "IN" clause, a search may be confined to document titles, authors' names, abstracts or other document subdivisions. When PARAGRAPH or SENTENCE is specified after IN, the search logic defined in the "logical expression" is applied separately to each paragraph (sentence) in each

document searched. Hence if the user requests

```
FIND  'TERM A' * 'TERM B' IN SENTENCE,
```

the strings 'TERM A' and 'TERM B' must both occur in the same sentence in order for a document to respond. A document which contains both 'TERM A' and 'TERM B', but not in the same sentence will not be retrieved by this request. If, however, the user requests

```
FIND  'TERM A' + 'TERM B' IN SENTENCE,
```

any document which contains either 'TERM A' or 'TERM B' anywhere between sentence boundaries will respond. The effect of the "IN" clause in this case would simply be to restrict the search to TEXT (ABSTRACT, BODY, NOTES), REFERENCES, and COMMENTS sections since these are the only sections which contain SENTENCE subdivisions.

Only one context name from the list in Figure 2.1 may be specified in an "IN" clause. If no "IN" clause is supplied, the default is ARTICLE.

3.1.1 Examples of Valid FIND Instructions

```
FIND 'KWIC' + 'KEY' * 'WORD' * 'CONTEXT' IN TITLE

FIND     'FIND'

FIND ('ON-LINE'+'REAL TIME'+'TIME SHAR')*('COMPUT'+'PROCESS'
     +'SYSTEM')
```

This last example would restrict retrieval to those documents employing the spelling conventions shown, it would not retrieve articles referring to "REAL-TIME SYSTEMS" or to "REALTIME SYSTEMS". Two other formulations that would avoid this restriction are:

```
FIND ('ON'*'LINE' + 'REAL'*'TIME' + 'TIME'*'SHAR')*('COMPUT'
     + 'PROCESS' + 'SYSTEM')
```

and

```
FIND ('ON'*'LINE' + 'TIME'*('REAL' + 'SHAR')) * ('COMPUT'
     + 'PROCESS' + 'SYSTEM')
```

3.1.2 Examples of Invalid FIND Instructions

```
FIND RETRIEVAL
     (no apostrophes around "RETRIEVAL")

FIND 'BONE MARROW' AND 'TRANSPLANT'
     ("*" is the required symbol for logical "AND")

FIND'ZIPF' IN AUTHOR
     (blank must follow "FIND")

FIND ('TIME' * ('REAL' + 'SHAR') * 'COMPUT'
     ( a "(" occurs without a corresponding ")" )
```

3.2 PRINT Instruction

The PRINT instruction indicates to the control program which sections of a responding document are to be printed after a search. Its format is:

PRINT < context list >

where the "context list" consists of one or more context names (Figure 2.1) separated by blanks or commas. Any number of context names may be specified and in any order.

Two asterisks (**) are placed in the margin of the printed copy beside every line which contains any search string specified in the associated FIND instruction.

Normally, no context unit will be printed more than once. Two or more equivalent or overlapping context names may be specified; however, in such cases

the most general name in the hierarchy will be selected for printing, and the other related terms will be ignored. For example, if both SENTENCE and PARAGRAPH are selected, the result will be the same as if PARAGRAPH alone had been requested. Similarly, the following PRINT statements are all equivalent since TEXT is composed of the three sections ABSTRACT, BODY and NOTES:

```
PRINT  TEXT

PRINT  TEXT, ABSTRACT

PRINT  ABSTRACT, BODY, NOTES.
```

Because of the extent to which the user can control the contexts in which searching and printing take place, it has been necessary to establish conventions for handling a number of situations in which the intended action is not clearly defined. For example, what is the meaning of the request:

```
FIND 'TERM A' * 'TERM B' IN TEXT

PRINT  SENTENCE                    ?
```

In such cases the nature of the printed output depends upon the context of the FIND instruction as well as those in the PRINT request.

The rules which govern printing under various conditions are given below. Regardless of the order in which contexts are given in the PRINT statement, they are processed in the order shown. Throughout this discussion, the phrase "major context unit" refers to any of the following: DOCUMENT, ARTICLE, DATA, INDEX, KEYS, TEXT, ABSTRACT, BODY, NOTES, REFERENCES or COMMENTS; "minor context unit" refers to SENTENCE or PARAGRAPH.

DOCUMENT, ARTICLE

The entire document is printed, and all other printing requests are ignored.

DATA

The complete bibliographic data section is printed, and all separate requests for individual bibliographic items, such as TITLE or AUTHOR, are ignored.

AUTHOR, TITLE, SOURCE, DATE, PAGES, MISC.

Selected items are printed in the order shown.

INDEX, KEYS, ABSTRACT

These major context units are printed in the order shown. Note that a request to print ABSTRACT always produces a complete copy of the abstract. As explained below, this is not true of other major document sections which contain paragraph and sentence subdivisions.

PARAGRAPH, SENTENCE

The processing of print requests for PARAGRAPH and SENTENCE depends both upon what other contexts are to be printed and upon what context has been specified in the FIND instruction. Print requests for PARAGRAPH or SENTENCE do not affect the printing of DOCUMENT, ARTICLE, or ABSTRACT; but they supersede requests for TEXT, BODY, NOTES, REFERENCES and COMMENTS. If both PARAGRAPH and SENTENCE are requested, PARAGRAPH is selected.

The following paragraphs describe the interaction of the instruction "PRINT PARAGRAPH" with the various contexts that may be selected in the FIND instruction. Similar remarks apply to "PRINT SENTENCE".

If the FIND context is DATA, INDEX or KEYS (major context units which do not contain paragraphs) no output is produced.

If the FIND context is any bibliographic subdivision (TITLE, AUTHOR, etc.), the full bibliographic citation is printed.

If the FIND context is any other major context unit, all paragraphs are printed which lie within that context unit and which contain any of the search strings requested in the FIND statement, regardless of the Boolean relationships that may have been specified there.

If the FIND context is PARAGRAPH or SENTENCE and the PRINT context is PARAGRAPH, only those paragraphs which completely satisfy the search request, including the Boolean relationships, or which contain sentences that do so are printed. Similarly, if the FIND and PRINT contexts are both SENTENCE, only sentences which satisfy the search request are printed.

If the FIND context is PARAGRAPH and the PRINT context is SENTENCE, then from those paragraphs that satisfy the search request every sentence containing any of the requested search strings is printed.

TEXT, BODY, NOTES, REFERENCES, COMMENTS

In the absence of print requests for SENTENCE or PARAGRAPH, these major context units are printed in the order shown. If TEXT is specified, separate requests for BODY and NOTES are ignored.

At the present time, only the control routine can initiate a PRINT instruction: it does this as part of its preparation for a search. In this mode, the system types the question, "PRINT?", and the user responds by typing a context list. A later version of the program will allow the user to request directly the printing of specified documents or document subsections and, in particular, of documents which responded in any of several previous searches.

3.3 Planned Extensions

Several new features will be added to the basic system shortly in order to increase the user's ability to control a search. FIND instructions

will be numbered sequentially, and the text of each question will be saved along with a list of documents which responded to that question. It will be possible to specify that a new search is to be restricted to the particular documents specified or to the set of documents retrieved in any previous search. It will also be possible to combine the results of several previous questions, using logical AND, OR and NOT, in order to produce new document sets which may be searched subsequently.

One other feature planned for the system is the COMMENT facility. This command is visualized as an underlining and note-taking facility which will allow the user to enter any remarks he may wish to make for storage directly with the text of a document. In the initial implementation of the system, disk space will be reserved at the end of each document for comments. These comments will be entered along with some user identification and will become a semi-permanent, searchable part of the text. Eventually, with the aid of graphics terminals, entry of some comments may be performed very much like underlining in a book; and it may be possible to transfer parts of the original text into a user's private file for later reference and use.

4.0 Error Messages

This section contains a list of retrieval system error messages and their interpretations. Whenever possible, the system places an up-arrow under the input character position at which the error was detected.

MESSAGE	NOTES
Invalid Character or Keyword	An undefined (possibly misspelled) keyword or context name or an undefined operator symbol has been detected.
Invalid Successor	A search request is not well formed. This message results from syntactic errors such as two successive operators, two successive search terms, the keyword "FIND" followed immediately by an IN clause, etc.
Missing " ' "	The total number of apostrophes in a search expression is odd, indicating an error in specifying some search term.
Unbalanced ( )'s	The total numbers of right and left parentheses in a search expression are not equal, or a right parenthesis has been detected before a corresponding left parenthesis.
Too Many ( ) Levels	Parenthesized quantities are nested to a depth greater than 13.
Too Many Terms or ( )'s	The total number of search terms and parenthesized quantities in a search expression exceeds 30.
Too Many Disjunctions	More than 256 terms and parenthesized expressions are joined together by "+" (OR). There is no corresponding restriction on the use of "*" (AND). In any event, other limits should be exceeded before this one.
STRTXT Table Full	Storage capacity for search terms has been exceeded. The total number of characters allowed is approximately equal to 100, depending upon the lengths of individual terms.
Stack Full	Insufficient temporary storage space available for the search scheduling procedure. User should simplify the form of the search request, if possible, or resubmit as two or more separate inquiries.
System	A programming or data base formatting error has been detected. Correction by system maintenance personnel is required. Temporarily, the user may be able to complete his search by removing or changing restrictions on the contexts to be searched or printed.

PART II. Technical Description

1.0 Introduction

The retrieval program described in this report is the initial implementation of an experimental system designed to support the development of effective algorithms for retrieving information from a large variety of data bases, and especially from data bases which have very little inherent structure. It is felt that direct sequential scanning of the data will necessarily play some part in the operation of such a general retrieval system, and an immediate goal of this program is to evaluate quantitatively various strategies for scheduling a complicated search request in this environment. Section 4 discusses in detail the facilities provided for studying this problem.

Another immediate goal is to study in a controlled environment the interaction between the system and a group of motivated users who are not necessarily experts either in computer techniques or in the subject matter of the data base.*

These research goals, together with the anticipated characteristics of the computer system to be used, have lead to the following design specifications and constraints:

1. Searching is to be performed by means of a sequential scan through the complete text of the data base.
2. The user should have very general control over the text of the search terms, the logical structure of the search request, and the contexts to be searched and printed.

---

*A collection containing the full text of 65 technical articles in the field of information retrieval has been obtained for initial experimentation.

3. It should be possible to search for co-occurrences of terms within sentences and paragraphs (or comparable subdivisions in a non-textual data base) as well as individual items of bibliographic data and larger document subdivisions.
4. The user should be given some feedback, in the form of printed output, immediately after a document is retrieved. (This is probably impractical in a large system where a single inquiry may retrieve several hundred citations. The more common practice is to report to the user the number of citations retrieved by a search and allow him to modify his inquiry, request printed output or take some other action.)
5. A special symbol is to be placed in the margin of each line of printed output which contains any of the requested search terms.
6. No part of any document is to be printed more than once in response to any given search request, even though the user may specify overlapping print contexts such as ABSTRACT and SENTENCE or ABSTRACT and TEXT. (See Part I of this report.)
7. System resources, especially memory, will be quite limited. The program must operate from a memory containing 4096,* 16-bit words. During execution of a search, about half the memory should be reserved for text.
8. Data will be accessed from the disk by sectors, where each sector contains about 500 words (1000 characters) and each track contains

---

*8192 words are available in the present configuration, but the original design was for 4096, and the program can run in that space.

8 sectors. Any number of sectors may be requested in a single read instruction.

9. Disk I/O is to be minimized--whenever possible a given body of text should be read from disk only once per inquiry.

Because this program is experimental and because it was written for a newly-designed minicomputer (the Burroughs D-Machine) before the actual hardware became available and before the configuration to be installed had been completely determined, it inevitably contains:

1. Some facilities whose value is unknown.
2. Some facilities designed to assist in testing and revising the main program, notably the DUMP and CHANGE routines. These procedures normally would not be included in a "production" version of the program which was to be used for retrieval experiments.
3. Some implementation features which should be reexamined in light of the conditions which actually exist. In particular, it was felt originally that memory space would be the most critical resource, and several design decisions were made in the interest of conserving memory.

The design configuration for which the retrieval program was written consists of a microprogrammable Burroughs D-Machine minicomputer with 1024, 64-bit words of microstore and 4096, 16-bit words of main memory. Peripherals include a card reader, a line printer, and a disk with a storage capacity of about 25 million bits. The retrieval program is written in an assembly language called the S-Language. The S-Language and its assembler are described in detail in [1] and [2]; but some of its features, which are essential to an understanding of the present report, will be reviewed here.

The language consists primarily of "Word Instructions", which perform

standard arithmetic, logical, and control functions and "String Instructions" which perform complicated manipulations and searches involving character strings. Sixty-four "software registers" are reserved in main memory for supplying the various pointers, characters, counters and transfer addresses required by the string instructions. The three string instructions of interest here are FIND, COMPARE, and SEARCHF.

FIND searches character string S1 for an occurrence of string S2 until either end character K1 is detected in S1 (failure) or end character K2 is detected in S2 (success). After execution, the pointers associated with S1 and S2 may be independently set at their initial or final positions or at the first character on either side of the final position. The number of characters examined in S1 is stored automatically in one of the counter registers. Finally, control is passed to the next instruction in the program or to one of two, independently specified, alternative success or failure transfer addresses.

COMPARE operates much like FIND except that the two strings are compared directly, and the comparison terminates whenever an end character or a mismatch is detected. Processing of pointers and transfer addresses after execution is similar to that for the FIND instruction.

SEARCHF searches forward through the character string S1 until any of three specified key characters is located. Again, pointer manipulation is performed, a character count is saved and independent transfer addresses may be specified for each of the three keys.

The remainder of this report describes the operating details of the retrieval control program. Section 2 deals with the structure and searching of certain control tables. Sections 3 and 4 describe the parsing of a search request and the scheduling of the search. Section 5 discusses the decoding of a PRINT instruction. Sections 6 and 7 explain the actual control of searching

and printing and the interaction between the two. Sections 8-10 deal with the possible implementation of a negative search request (FIND 'TERM A' BUT NOT 'TERM B'), with the effects of certain system errors (mainly errors in the data base), and with those debugging facilities which are incorporated into the present version of the program. Flow charts and register assignments are given in the appendices.

Throughout this report, hexadecimal character strings will be written between colons as, for example, in :80AD:, which represents the bit string '1000 0000 1010 1101'.

2.0 Table Searching

In order to conserve memory space and to take advantage of the specialized text-searching features of the S-Language, the system tables INSTKEY, RESKEY and CTXT have been designed for access by sequential search. As a result, some frequently-occurring searches can be performed by means of a single string instruction and others by a standard short procedure (TSRCH) containing only five instructions. Many of the required register-loading operations need be performed only once in preparation for an arbitrary number of similar searches.

Three special characters, :81:, :82:, and :80: are used in constructing these tables. The :81: and :82: mark the beginning and end, respectively, of each search key, and are used to guarantee an exact match between the input string and the table entry. The :80: identifies the end of the table. Data associated with a search key begins in the high-order byte of the first word following the :82: and continues for as many bytes as necessary. The fill character :00: is used as required to assure word-boundary alignment for the data entries.

A typical table organized for sequential searching is INSTKEY, which contains the keywords FIND, PRINT*, DUMP*, and CHANGE*, used to identify input instructions. Following are the first few entries in this table:

```
:81:"FIND":82:A(FIND PROCESSING):81:"PRINT":8200:A(PRINT
PROCESSING) . . .
```

where hexadecimal digits (4 bits each) are contained between colons, alphabetic character strings (8 bits/character) are enclosed in quotation marks and A(X)

---

*PRINT is a legal input even though it causes no processing in the present implementation. DUMP and CHANGE are system commands to be explained in Section 10.

(one full 16-bit word, aligned on a word boundary) represents a transfer address to the appropriate instruction decoding procedure.

In order to identify an input instruction, the program first inserts the character :81: ahead of the first non-blank character on the input line. It then calls TSRCH, which executes a FIND instruction in an attempt to locate the input keyword in the INSTKEY table. Searching stops whenever a blank is located in the input line (instruction keywords must be followed by at least one blank) or when the character :80: (end-of-table) is encountered in the table. If an :80: is detected, control is transferred to an error handling routine; otherwise, the Data Pointer (pointing into the table) is left with the byte address of the first unmatched character. The program next attempts to verify by means of a COMPARE instruction that that first unmatched character is :82:. If the comparison fails, control is transferred directly back to the FIND instruction described above, and the search resumes where it left off. If the comparison succeeds, the Data Pointer is left pointing two characters beyond the :82:, i.e., to either byte of the word containing the desired transfer address; and control is returned to the calling program, which uses the contents of the table pointer to access the required data.

In addition to INSTKEY, RESKEY and CTXT, the table, CHARS, is also entered by sequential search; but it is only used to identify a character or character type according to its position in the table, as recorded automatically in Counter Register 0 at the completion of the search.

It is usually possible to modify these tables simply by deleting unwanted entries or adding new ones at any convenient position. In general, frequently-used entries should be stored ahead of those which are less commonly accessed.

3.0 FIND Statement Processing

Conceptually, search requests transmitted to the system by means of a FIND statement can be arbitrarily complicated Boolean expressions whose "variables" are character strings to be located in the data base. Search terms are separated by "+" (logical OR) or "*" (logical AND) and may be grouped in any desired way by means of parentheses. In the absence of parentheses, the operator AND is considered to be dominant over the operator OR, i.e.,

X + Y * Z

is equivalent to

X + (Y*Z).

Negation is not allowed in the current version of the system; although, as explained in Section 8 below, provision has been made for its later implementation.

In order to control the progress of a search, it is necessary to convert the user's request into a form which preserves the logical structure of the original but which can be manipulated more conveniently. This conversion is accomplished with a single, serial, left-to-right scan of the input line, using a procedure based on a system of internally generated tags similar to that employed in the "Decision Module Compiler" (DMC) [3]. In the DMC, these tags are treated effectively as statement labels to assist in code generation. In the system under discussion here, they are used in a corresponding way to construct a table for controlling the retrieval operation and for determining the order in which search terms are considered. The construction and use of such a table and the search scheduling algorithms, to be discussed in Section 4, are believed to be original.

3.1 Tag Structure and Meaning

A syntactically correct search request contains up to six kinds of elements in addition to the keyword FIND:

1) Search terms: character strings enclosed between apostrophes
2) Operators: "+" and "*"
3) Left Parentheses
4) Right Parentheses
5) End Symbol: carriage return or the character string "IN_", where "_" represents a blank
6) Blanks not included within search terms.

The term "entity" will be used to refer to two types of expressions composed of these elements:

1) Search terms and
2) Character strings beginning with a left parenthesis and ending with a corresponding right parenthesis.

Note that one entity may be contained within another as in

('A' + ('B' + ('C' * 'D'))),

which contains seven entities, (four search terms and three parenthesized expressions). Nevertheless, any valid search request may be regarded as a series of alternating entities and operators, beginning and ending with an entity, where each entity may in turn contain a similar alternating series.

In building the required data structure, an internally generated tag is assigned to each entity in the search request. Each tag is stored in one full word of memory (16 bits) and contains two fields: the level field (4 bits) and the disjunction field (8 bits), (see Figure 3.1). The level field reflects the depth to which an entity is nested in parentheses; the disjunction field indicates the relationship between the current term and the one immediately preceding it at the same level. The high-order bit of each tag is reserved

for use by the search control routines; the other three bits are available for expansion, e.g., to accommodate the operator "NOT".

* RESERVED FOR SEARCH CONTROL ROUTINES

Figure 3.1. Tag Structure

In parsing a search request, the entire expression is treated for internal purposes as if it were enclosed in parentheses and is assigned the tag :0100:. This is the only level 1 entity that ever occurs.

The scanning procedure next identifies the first entity of the input statement and assigns to it the tag :0200:. If another entity exists at level 2, it is assigned either the tag :0200: or the tag :0201: depending upon whether it is joined to the first entity by "AND" or "OR". As the processing continues, all related entities at a given level joined by "AND" receive identical tags and those joined by "OR" receive tags with increasing disjunction values. When a left parenthesis is encountered, the quantity it represents is assigned the appropriate tag, processing at the present level (L_i) is suspended, and a new series of tags at the next higher level (L_{i+1}) is initiated for the entities within the parentheses. When the corresponding right parenthesis is encountered, level L_{i+1} processing is terminated and level L_i processing is resumed. No overflow from one tag field to another is ever permitted, and attempts to "OR" together more than 256 entities or to nest parentheses to a depth greater than 13 result in error messages and termination of the parsing procedure.

As an illustration of the tag assignment system, consider the input expression

$$E * C *({}_1D + ({}_2({}_3F)_3)_2 * Y * A *({}_4B*Z)_4 + G*({}_5H+I*J+K)_5)_1 + L * M \quad (3.1)$$

For notational convenience in this example and those which follow, search terms will be represented by upper case alphabetic characters and no apostrophes will be shown. Subscripts will be attached to left and right parentheses for the purpose of identifying "mates" and for use in referring to parenthesized quantities.

As explained previously, the entire expression would be assigned the tag :0100:. At level 2, the expression has the form

$$E * C *({}_1\text{---})_1 + L * M,$$

and the appropriate tags are :0200: for E, C, and $({}_1\text{---})_1$ and :0201: for L and M. Entities in the following subexpression receive level 3 tags:

$$D +({}_2\text{---})_2 * Y * A * ({}_4\text{---})_4 + G *({}_5\text{---})_5 \ .$$

As the scan proceeds from left to right, level 4 is "entered" three times, once for each of the following subexpressions:

$({}_3\text{---})_3$

$B * Z$

$H + I * J + K$.

At each entry, the level 4 assignment procedure is reinitialized so that tags are assigned as shown in Table 3.1. Because of the order in which tags are assigned and stored, and because related tags are joined together by a system of pointers (to be discussed in Section 3.2), no ambiguity results from assigning the tag :0400: to four different entities in three different level 4

entries. The complete list of tag assignments for this example is shown in Table 3.2.

Table 3.1. Level Four Tag Assignments

	Entity	Tag
Entry 1	$(_3\text{---})_3$	:0400:
Entry 2	B	:0400:
	Z	:0400:
Entry 3	H	:0400:
	I	:0401:
	J	:0401:
	K	:0402:

Table 3.2. Tag Assignments for Expression 3.1

Entity	Tag
1. Complete Expression	:0100:
2. E	:0200:
3. C	:0200:
4. $(_1$---$)_1$	:0200:
5. D	:0300:
6. $(_2$---$)_2$	:0301:
7. $(_3$---$)_3$	:0400:
8. F	:0500:
9. Y	:0301:
10. A	:0301:
11. $(_4$---$)_4$	:0301:
12. B	:0400:
13. Z	:0400:
14. G	:0302:
15. $(_5$---$)_5$	:0302:
16. H	:0400:
17. I	:0401:
18. J	:0401:
19. K	:0402:
20. L	:0201:
21. M	:0201:

By reversing the rules for assigning tags, one can reconstruct the form of a search request from the list of assigned tags, although that operation is not required in the retrieval program.

3.2 The TAG Table

For purposes of search scheduling and monitoring, tags are stored together with other definition and control information in a table called TAGS. The first two words of the table contain the word addresses of the first and last available locations in the table. Following this information, each data entry consists of four words:

WORD 1.	Tag
WORD 2.	STRTXT Address
WORD 3.	"Success" Pointer
WORD 4.	"Failure" Pointer.

If the tag stored in Word 1 represents a parenthesized expression, then Word 2 is set to zero. If the tag represents a search term, however, then Word 2 contains the byte address of the first character of the term as stored in the STRTXT Table (Section 3.3). The significance of "success" and "failure" in connection with the last two words of a Tag Table entry will be explained in Section 4. For now it is sufficient to note that these two words contain a system of pointers used to control the progress of a search. This system is constructed in two phases, the first performed by the parsing routine and the second by a later search scheduling procedure.

The end of an expression in the Tag Table is identified by a zero in Word 1 of the first unused entry.

Each time a new level of parentheses is entered, the parsing routine constructs a chain of pointers linking all entities at that particular level

and entry. One of these pointers occupies Word 3 of each entry and points to Word 1 of its successor. A pointer value of zero identifies the last element on a chain. The first element on a chain is stored in Word 4 of the Tag Table entry for the parenthesized quantity itself, and it points to the Tag Table entry for the first entity inside the parentheses.

In addition to constructing a system of linked lists in the Tag Table, the parsing procedure can store in Word 4 of the entry for each search term selected information for use by the scheduling routines. At present, the length of the search term in bytes is stored in this location.

The complete Tag Table for Expression 3.1, as it would appear at the end of the parsing procedure, is shown in Table 3.3. In this table, A(X) represents the STRTXT address of search term "X"; and L(X) represents the length of term "X". For the purpose of illustrating pointers, Tag Table entries are numbered "Tn", where n is an integer; and the four words within an entry are labelled A, B, C and D. Hence, the notation "T6A" refers to the first word of Tag Table entry 6.

Table 3.3. Tag Table Entries for Expression 3.1 after Parsing Procedure

	A	B	C	D
T1	:0100:	0	0	T2A
T2	:0200:	A(E)	T3A	L(E)
T3	:0200:	A(C)	T4A	L(C)
T4	:0200:	0	T20A	T5A
T5	:0300:	A(D)	T6A	L(D)
T6	:0301:	0	T9A	T7A
T7	:0400:	0	0	T8A
T8	:0500:	A(F)	0	L(F)
T9	:0301:	A(Y)	T10A	L(Y)
T10	:0301:	A(A)	T11A	L(A)
T11	:0301:	0	T14A	T12A
T12	:0400:	A(B)	T13A	L(B)
T13	:0400:	A(Z)	0	L(Z)
T14	:0302:	A(G)	T15A	L(G)
T15	:0302:	0	0	T16A
T16	:0400:	A(H)	T17A	L(H)
T17	:0401:	A(I)	T18A	L(I)
T18	:0401:	A(J)	T19A	L(J)
T19	:0402:	A(K)	0	L(K)
T20	:0201:	A(L)	T21A	L(L)
T21	:0201:	A(M)	0	L(M)
T22	:0000:	---	---	---

3.3 The STRTXT Table

Character strings specified by the user as search terms are stored in the STRTXT table. Search terms are stored in one continuous string with the character :80: serving as a separator and as a stop character for search operations. The first two words of the table contain, respectively, the address of the first available byte and the address of the last byte reserved for data storage.

3.4 Processing Procedures

3.4.1 Tag Assignment -- The Level Table

Processing of the current input element, including tag assignment, is carried on by means of the two variables, CTAG and LINK, and a thirteen-level stack (LOGICLVL) to be called the Level Table. CTAG contains the tag to be assigned to the next entity encountered at the present level and entry. LINK contains the address of the link field (Word 3) of the last Tag Table entity at the present level and entry. The Level Table provides a means of restoring the values of CTAG and LINK that were effective at a particular level when processing at that level was suspended by the detection of a left parenthesis. The actions taken by the processing procedure for each kind of input element are summarized in Table 3.4.

Table 3.4. Parsing Procedures

INPUT ELEMENT	ACTION
Left Parenthesis	1. Create Tag Table entry for parenthesized expression. 2. Enter address of new Tag Table entry at location shown in LINK. 3. Enter CTAG and LINK on top of Level Table stack. 4. Generate new CTAG by adding 1 to previous value of level field and clearing disjunction field.
Right Parenthesis	Restore values of CTAG and LINK from top of Level Table stack.
Search Term	1. Create Tag Table entry for search term. 2. Enter address of new Tag Table entry at the location shown in LINK. 3. Move text of search term to STRTXT Table.
Operator "+"	Increment disjunction field in CTAG.
Operator "*"	Continue (no action required)
Carriage Return	Terminate Tag Table with :0000: in first unused tag field.
"IN "	Stop parsing logical expression; prepare to process an "IN" clause.
Blank	Skip.

3.4.2 Search Context Restriction -- The "IN" Clause

After the terminator "IN_" has been identified, the parsing routine locates the requested context name on the input line, performs a standard (sequential) look-up to locate the term in the CTXT table, and moves the appropriate context delimiters to Character Registers 0 and 1, which are reserved for this purpose.

Context delimiters are located in the low-order bytes of the first two words following the keyword end symbol (:82:). For example, :D3: and :D4: are the beginning and end delimiters, respectively, for TITLE, which has the following entry in the CTXT Table:

. . . :81:"TITLE":8200::8CD3::8CD4: . . . .

3.4.3 Syntax Checking

Syntax checking in the FIND decode routine consists mainly of identifying each new element in the input stream and determining whether or not this element is a legal successor for the previous element.

Consider first the rules of succession, as shown in Table 3.5. Each line in the table consists of four parts: an input element type, E; its Identification Code; a list of elements which may legally follow E; and a Legal Successor Code. The Identification Code designates a particular bit assigned to E from a 16-bit computer word. The Legal Successor Code is simply the disjunction of the Identification Codes for those elements which may legally follow E.

In order to test for legal succession using this system of codes, the parsing routine need only "AND" together the Legal Successor Code of one element and the Identification Code of its successor. If the result is non-zero the succession is legal; otherwise it is not.

In the present implementation, element identification and validity testing are accomplished by means of two tables, CHARS and LEGALTAB. CHARS

Table 3.5. Legal Successors

ELEMENT	IDENTIFICATION CODE	LEGAL SUCCESSOR							
		LEFT PAREN.	RIGHT PAREN.	OPERATOR	CARRIAGE RETURN	SEARCH TERM (APOSTROPHE)	CONTEXT NAME	'IN_'	LEGAL SUCCESSOR CODE
LEFT PAREN.	:0001:	X				X			:0011:
RIGHT PAREN.	:0002:		X	X	X			X	:004E:
OPERATOR	:0004:	X				X			:0011:
CARRIAGE RETURN	:0008:								:0000:
SEARCH TERM (APOSTROPHE)	:0010:		X	X	X			X	:004E:
CONTEXT NAME	:0020:				X				:0008:
'IN_'	:0040:						X		:0020:

consists of a list of initial characters by which elements may be identified:

_,(,),+,*,',:0D0D:,A,B,C,...,X,Y,Z,0,1,2,...,7,8,9,:8080:

where "_" represents a blank, :0D: is a carriage return and :80: marks the end of the table. LEGALTAB contains one three-word entry for each type of element:

WORD 1. Transfer address to processing routine for this element.

WORD 2. Identification Code for this element.

WORD 3. Legal Successor Code for this element.

When the first character of a new element has been found, an attempt is made, using a Search Forward (SEARCHF) instruction, to locate either that first character or an :80: in the CHARS Table. After execution of this instruction, the number of characters examined (minus 1) is stored automatically in Counter Register 0 and can be used to identify the character. If the symbol is a "special character" ($1 \leq CTR0 \leq 6$), then CTR0 is used directly as an index into LEGALTAB. Otherwise further searching in other tables must be performed to identify the new element as a legal context name, the word "IN" or an illegal alphanumeric string.

After the new element has been identified, its Identification Code is compared with the Legal Successor Code for the previous element and, if the succession is legal, control is transferred to the appropriate processing routine. All data necessary for this procedure, including the transfer address, is obtained from LEGALTAB.

Other error checking procedures consist mainly of testing for overflow in the various tables and tag fields. The Level Field in the variable CTAG provides a convenient counter for detecting unmatched parentheses: a right parenthesis detected while the level of CTAG < 3 is unmatched, and some left parenthesis is unmatched if an end symbol is detected when the level of CTAG $\neq 2$.

4.0 Search Scheduling

4.1 Scheduling Criteria

As discussed previously, an important reason for building the present retrieval system is to investigate algorithms for predicting the most efficient order of search among several terms in a complicated inquiry. While the experimental system employs a direct sequential search to locate responding documents, the results of this study should be applicable in an inverted file system as well, where scheduling procedures can be used to reduce term coordination time by "controlling" the lengths of the intermediate postings lists which develop during the coordination procedure.

The basic idea is that in a search for 'TERM A' and 'TERM B' together in a restricted context, whichever term is less likely to occur should be sought first, since then more context units can be rejected after a single scan because they do not contain that first term. A corresponding statement applies to a search for 'TERM A' or 'TERM B'.

The problem lies in finding some reasonably reliable yet simple way of determining relative probabilities of occurrence for individual character strings and for the arbitrary combinations of strings which may occur in complicated search requests.

This problem would be partially solved if one knew in advance the probability of occurrence of each legal search term in any given context. This condition is approached in some inverted file systems, where one may know how many citations are associated with each search term before the required processing begins. However, these frequency counts are usually associated only with complete individual words. The construction, maintenance and use of frequency tables for word fragments or arbitrary character strings would be very difficult, or perhaps impossible, even for a small data base.

Alternative indicators of frequency of occurrence to be investigated include the number of characters in the input string, the least common bigram or trigram (two or three character sequence) in the input string (i.e., the bigram or trigram in the input string which occurs least frequently in the data base), and the least common initial bigram or trigram. The easiest of these systems to implement and the one which is used in the current version of the program is one based on the length of the input string. An analysis of word frequencies in thirty-three documents in our experimental data base has shown that with the exception of lengths one and six, the frequency of occurrence of a word is a decreasing function of its length. It is reasonable to expect a similar trend among arbitrary character strings.

The second part of the problem--the scheduling of arbitrary combinations of strings and parenthesized expressions which may occur in a complicated search request--is a subject for experimentation. The rules which have been adapted initially will be explained below.

4.2 Changes in Scheduling Procedures

Because the search scheduling procedure is to be a topic for experimentation, the retrieval program has been designed to permit easy substitution of one algorithm for another. After completion of the FIND decoding process, the TAG Table as it appears in Table 3.3 is passed to the subroutine SHALG for scheduling. Changing the scheduling algorithm consists of providing a new version of SHALG.

Two types of scheduling changes are possible: changes in the method for determining the relative frequencies of the search terms and changes in policy concerning the scheduling of subexpressions within a complicated search request. Changes of the first type can be accomplished simply by adding to the

existing routine a preprocessing step which changes the contents of Word 4 of the Tag Table entry for each search term. This word is reserved for an index which reflects the expected relative frequency of the term. In the present implementation, large index values correspond to low frequencies, and the index in use is the length of the term in bytes. If scheduling were to be based, instead, on least common bigrams, one could list all bigrams in the data base according to frequency from most frequent to least, determine which bigram in the input string lay closest to the end of the list, and use its position in the list as the index for the term. No other changes in procedure would be required.

Changes of the second type, basic scheduling policy, require substitution of a new version of SHALG.

4.3 Success and Failure Linkage

Search scheduling and control are accomplished by means of two systems of pointers, called success and failure links, in columns C and D of the Tag Table. Consider the search request

FIND A*B*C + D*E*F, (4.1)

and suppose that the search terms were processed in their original order from left to right. The first step of the procedure would be to scan the text for term A. If A were found, then a search for B would be conducted. However, if the search for A failed, searching for either B or C would be unnecessary, and processing could continue with term D. Thus, the success link for A would be B; and the failure link for A would be D. Similarly, the success link for B would be C, and the failure link for B would be D. The complete system of success and failure pointers for this example is shown in Table 4.1. Notice that each of a series of entities connected by the operator "*" has the same

failure link while each of a series of entities joined by the operator "+" has the same success link. Eventually every path through this structure leads to the condition of overall success or failure for the search.

The process of constructing this linkage is referred to as search scheduling.

Table 4.1. Success and Failure Linkage for Expression 4.1

TERM	SUCCESS LINK	FAILURE LINK
A	B	D
B	C	D
C	SUCCESS	D
D	E	FAILURE
E	F	FAILURE
F	SUCCESS	FAILURE

4.4 Final Organization of Tag Table

The following conventions are employed by the retrieval program in constructing success/failure linkage in the Tag Table:

1. Both success and failure pointers for a given term point to the Tag Table success column for the next term to be processed.
2. Ultimate success or failure is indicated by the entry :FFFF: in the appropriate link.
3. Both pointers associated with the Tag Table entry for a parenthesized expression point to the first entity to be processed inside the parentheses. The first entry in the table, which represents the entire search request, obeys this rule except that its success pointer is set to :0000:.

4. The success and failure links out of a parenthesized expression are recorded with the last term(s) to be processed inside the parentheses.

Recall that after the parsing procedure, all entities on a given linked list in the tag table which are logically connected by the operator "*" have the same tag, and entities joined by "+" have tags with the same level value but with different values in the disjunction field. On the assumption that progressively longer or more complicated expressions will be progressively less likely to occur, the scheduling procedure arranges the entities on each list for consideration in the following order:

1. Tags which represent strings only, but no parenthesized expressions (Group I tags) are considered first.
2. Tags in Group I which represent single strings are arranged in order from shortest string (most likely) to longest (least likely).
3. Tags in Group I which represent multiple strings are arranged in order from fewest strings to most.
4. Strings associated with each tag in step 3 are considered in order from longest to shortest.
5. Tags which represent parenthesized expressions and possibly individual strings as well (Group II tags) are arranged in order first from fewest strings to most and then from fewest parenthesized expressions to most.
6. Strings associated with each tag in step 5 are arranged in order from longest to shortest.
7. Parenthesized expressions associated with each tag in step 5 are arranged in order from fewest to most enclosed search strings.

Linkage between lists results from entry into or exit from a parenthesized expression.

These rules are illustrated in Table 4.2, which shows the complete Tag Table for the example of Expression 3.1, repeated for convenience as expression 4.2. The lengths assumed for the various terms are given in parentheses below the terms in 4.2. Pointers in the table are interpreted as for Table 3.3, where a pointer to "T21C" refers to column C of entry number 21.

$$\begin{array}{lccccccccccccccccccc} \text{FIND} & E & * & C & * & (_1 & D & + & (_2 & (_3 & F &)_3 &)_2 & * & Y & * & A & & \\ & (7) & & (2) & & & (4) & & & & (9) & & & & (7) & & (5) & & & \\ & * & (_4 & B & * & Z &)_4 & + & G & * & (_5 & H & + & I & * & J & + & & \\ & & & (6) & & (8) & & & (3) & & & (6) & & (4) & & (7) & & & \\ & K &)_5 &)_1 & + & L & * & M & & & & & & & & & & & \\ & (8) & & & & & (5) & & (8) & & & & & & & & & & \end{array} \qquad (4.2)$$

Table 4.2. Complete Tag Table for Expression 4.2

	A	B	C	D
T1	:0100:	0	:0000:	T21C
T2	:0200:	A(E)	T3C	:FFFF:
T3	:0200:	A(C)	T4C	:FFFF:
T4	:0200:	0	T5C	T5C
T5	:0300:	A(D)	:FFFF:	T14C
T6	:0301:	0	T7C	T7C
T7	:0400:	0	T8C	T8C
T8	:0500:	A(F)	T11C	:FFFF:
T9	:0301:	A(Y)	T10C	:FFFF:
T10	:0301:	A(A)	T6C	:FFFF:
T11	:0301:	0	T13C	T13C
T12	:0400:	A(B)	:FFFF:	:FFFF:
T13	:0400:	A(Z)	T12C	:FFFF:
T14	:0302:	A(G)	T15C	T9C
T15	:0302:	0	T16C	T16C
T16	:0400:	A(H)	:FFFF:	T19C
T17	:0401:	A(I)	:FFFF:	T9C
T18	:0401:	A(J)	T17C	T9C
T19	:0402:	A(K)	:FFFF:	T18C
T20	:0201:	A(L)	:FFFF:	T2C
T21	:0201:	A(M)	T20C	T2C
T22	:0000:	---	---	---

To use Table 4.2, consult first column D of entry T1, which contains a pointer to the first term to be located (M), then proceed as explained in the text.

4.5 Processing Procedures

Most of the processing in SHALG consists of selecting various groups of tags and sorting them into the desired order. After this process has been completed, the Tag Table appears much as it did upon entry to the routine except that the order of the elements on the various linked lists has been changed. It is now time to make the SUCCESS/FAILURE assignments.

Success and failure assignments proceed in a straightforward manner according to the principles in the previous section and the following rules:

1. The success link for the first entry in the Tag Table is set to :0000:; the failure link points to the first entity to be processed.
2. If the current entity is the last element on a chain having a particular tag, then SUCCESS means success for the chain; otherwise the success link points to the next element on the chain (which necessarily has the same tag as the current element).
3. If some other entry X, further down the chain has a different tag from that of the current entity, then the failure link points to the first such X; otherwise, FAILURE means failure for the chain.
4. By virtue of the way the chains are constructed, only one chain will ever occur at level 2, and success or failure on that chain implies success or failure for the search as a whole. (Success or failure for a higher level chain, on the other hand, implies

success or failure in satisfying the requirements of a parenthesized expression.)

5. When a tag representing a parenthesized expression is encountered, the success and failure links are determined in the standard way and saved in a stack along with the link to the next element on the current chain. Processing of the current chain is suspended, and success/failure assignments are completed for the terms inside the parenthesized expression. "Ultimate" success and failure links for the parenthesized quantity are obtained from the next lower level in the stack.
6. When all terms inside a parenthesized expression have been processed, the link is recovered from the top of the stack, and processing of the next lower level chain continues.
7. The process terminates when the end of the level 2 chain is detected.

5.0 PRINT Statement Processing

PRINT statement processing consists of locating the requested context names in the CTXT Table and marking the entries appropriately. The data portion of an entry in the CTXT Table consists of 4 bytes of the form

8X YY 8C ZZ

where YY and ZZ are the beginning and end delimiters, respectively, for the associated context; and X is either C or D depending on whether or not printing of the associated context has been requested. Therefore, the PRINT statement processor first "clears" the CTXT Table, replacing each X with C. It then isolates context names in the input line and changes the X's in the corresponding data fields from C's to D's.

6.0 Search Control

6.1 Detailed Data Base Structure

Figure 2.1 in Part I of this report describes the hierarchical structure of documents in the data base. Table 6.1 of this section lists the section names together with the hexadecimal start and end characters for each document subdivision. Note the new division, "DIRECTORY", which has been inserted ahead of "DOCUMENT". This is a non-searchable context directory used by the control programs to facilitate searching.

The arrangement of the various context sections and delimiters as they would appear in storage is illustrated in Figure 6.1. Three different types of context delimiters (corresponding to prefix characters "C", "D", and "E") are available.

6.1.1 C-Delimiters

C-delimiters are used to identify major sections of a document such as ABSTRACT, bibliographic DATA, etc. Each of these must occur once and only once in any given document, even if some of the sections they identify are absent. If, for example, a document contains no entries under INDEX or KEYS, then these sections should be represented by their delimiters alone, as follows:

... DCC2C3C4ECEE ... .

C-delimiters must appear within a document in increasing numerical order from C0 through C9.

6.1.2 D-Delimiters

D-delimiters identify individual items of bibliographic data, such as titles or authors' names, which are to be available for direct searching. They

Table 6.1. Context Delimiting Characters

Section Name	Context Delimiters (Hexadecimal)	
	Start	End
DIRECTORY	C0	C1
DOCUMENT, ARTICLE	C1	C9
DATA	C1	C2
AUTHOR	D1	D2
TITLE	D3	D4
SOURCE	D5	D6
DATE	D7	D8
PAGE, PAGES	D9	DA
MISC	DB	DC
INDEX	C2	C3
KEYS	C3	C4
TEXT	C4	C7
ABSTRACT	C4	C5
BODY	C5	C6
NOTES	C6	C7
REFERENCES, REFS	C7	C8
COMMENTS	C8	C9
PARAGRAPH	EC	EC
SENTENCE	EE	EE

00C0 ... (directory data -- 10 words) . . . ECEE880000000000

C1D1 ... (author) . . . D2D3 . . . (title) . . . D4D5 . . .

(source) . . . D6D7 . . . (date) . . . D8D9 . . . (pages) . .

. . . DADB . . . (miscellaneous bibliographic data) . . . DC

C2 . . . (index) . . . C3 . . . (keys) . . . C4ECEE . . . . .

(abstract) . . . EE . . . EE . . . EEEC . . . . . EEC5EC .

. . (body) . . . EE . . . EE . . . EEEC EEC6EC

. . . (notes) . . . EE . . . EE . . . EEEC . . . . . EE

C7EC . . . (references) . . . EE . . . EE . . . EEEC . . . . .

. . . EEC8EC . . . (comments) . . . EE . . . EE . . . EEEC . .

. EE . . . EEECC9

Figure 6.1. Document Organization in Storage

make it possible to search rapidly for all items by a particular author or from a particular journal, or from selected years. As with C-delimiters, each D-delimiter must occur once and only once within each document. However, strict numerical order need not be maintained, i.e., individual items of bibliographic information need not appear in any fixed order.

6.1.3 E-Delimiters

E-delimiters separate repeated elements of text, such as paragraphs and sentences, which are to be separately searchable. One sentence delimiter (EE) appears at the beginning and one at the end of each sentence in the text. Only one delimiter need appear between two sentences. Similar statements hold for the paragraph delimiter (EC). Each major document section which can contain E-type subdivisions (see Figure 6.1) must contain at least one paragraph symbol and one sentence symbol, even if no other text is present. Hence, if a certain document contained no reference list and no comments, these sections would appear as follows:

... C7EEECC8EEECC9 .

There is no theoretical restriction on the length of any document section or subsection in this system; however, as a matter of convenience for the current implementation, it is assumed that the entire bibliographic data section can be contained in core at once for searching. This limits that one section to a total length of approximately 4000 characters.

6.2 Directory and Control Data

The first line of Figure 6.1 shows several words of control

information stored at the beginning of a document between the delimiters C0 and C1. This information, which is generated when a document is added to the data base, consists mainly of pointers to the disk addresses at which the various major sections of the document begin. The control programs use these pointers to locate the beginnings of major sections of text without having to search sequentially from the beginning of the document. This directory can also be used to locate the "next" document in the file without reference to independent system tables, thus reducing requirements for core storage (a limited resource) or for disk access.

Table 6.2 shows the contents of the 10 words in the document directory. The first eight of these words contain the addresses of the disk sectors in which the eight major document sections begin. It is important to note that directory pointers do not indicate the exact location of the first character of a document section, but only the address of the disk sector which contains that first character. A sector is the smallest addressable unit of data on the disk and, in the present system, contains about 900 characters.

Directory words nine and ten contain sector addresses, respectively, for the current end of the document and for the end of the disk space reserved for the document. Typically several unused sectors are reserved at the end of a document for storing users' comments. When comments are present, they will constitute a searchable field and will affect the value of the end of text pointer.

The four words which follow the directory (see Figure 6.1) contain "stop" characters for certain searches which proceed in the reverse direction and empty space for use by the routines which format text for printing.

In addition to the 14 words of directory and control data stored with each document, twelve locations in core are permanently reserved for control

Table 6.2. Directory Contents

WORD 1 - Disk sector address of start of DATA section

WORD 2 - Disk sector address of start of INDEX section

WORD 3 - Disk sector address of start of KEYS section

WORD 4 - Disk sector address of start of ABSTRACT section

WORD 5 - Disk sector address of start of BODY section

WORD 6 - Disk sector address of start of NOTES section

WORD 7 - Disk sector address of start of REFERENCES section

WORD 8 - Disk sector address of start of COMMENTS section

WORD 9 - Disk sector address of the current end of text

WORD 10 - Disk sector address of the end of allocated disk space for this document

purposes: ten consecutive locations labelled CNTRL00--CNTRL09 immediately preceding the buffer space used for document text, and two other locations labelled DISKLIM1 and DISKLIM2, which contain, respectively, the disk sector addresses of the beginning and end of the data base. All disk space between these two addresses is assumed by the program to be allocated to document text, although it need not all be filled. The allocation of disk space to individual documents is controlled by directory information in the affected documents. No master disk directory is required by the retrieval program although it will probably become desirable to implement one at some later date.

The use of control words CNTRL00--09 is defined in Table 6.3. They provide a means for establishing correspondence between the physical locations of disk sectors in core and the disk sector addresses listed in the document directories.

Words CNTRL01 and CNTRL02 are loaded by the S-Level program before executing a disk read instruction. The remainder of the information is supplied by the microprogram as the read proceeds.

Table 6.3. Control Word Assignments

CNTRL00	-	Byte address of end-of-buffer character (:88:) supplied by microcode after disk read
CNTRL01	-	Disk address of first sector in core
CNTRL02	-	Byte address of first character from first sector read into S-Memory
CNTRL03	-	Byte address of first character from second sector read into S-Memory
.	.	.
.	.	.
.	.	.
CNTRL09	-	Byte address of first character from eighth sector read into S-Memory

6.3 Search Control Procedures

Several technical problems arise from the extreme variation which exists in the lengths of the context units to be considered and from the fixed, somewhat limited space available in memory. In some cases arbitrary design decisions, hopefully in accord with the specifications stated in Section 1, have been required; and these are always potentially subject to revision.

6.3.1 Search Types

First, consider the search request

FIND 'TERM A' * 'TERM B' + 'TERM C' * 'TERM D' IN ARTICLE, (6.1)

and suppose that the terms have been scheduled in the manner shown in Table 6.4.

Suppose further that some document, K, contains enough text to fill the available memory N times, where N > 1.

Table 6.4. Search Schedule for Expression 6.1

TERM	SUCCESS POINTER	FAILURE POINTER
START	:0000:	TERM A
TERM A	TERM B	TERM C
TERM B	SUCCESS	TERM C
TERM C	TERM D	FAILURE
TERM D	SUCCESS	FAILURE

The search scheduling procedure described in Section 4 requires an initial search for TERM A to continue until some occurrence of TERM A has been found or until the entire document has been scanned. If TERM A appears in Document K, then the process must be repeated for TERM B, etc., until the success or failure of the search has finally been established. This may require each memory load of text in Document K to be read from disk several times in the course of the search.

In order to avoid excessive disk handling, an alternative procedure has been adopted for searching major context sections, i.e., context units identified by C-type delimiters (Section 6.1.1). Each buffer is searched in turn for all terms in the search request and, when a string is located, the high order bit of its tag is set to "1" in the TAG Table. After a buffer has been completely processed in this way, the tags are used to determine whether or not the logical requirements of the search request have been satisfied. If not, more text is read from disk and the search continues for those strings not yet found.

Now consider the example of 6.1 and Table 6.4 with the search context changed to PARAGRAPH. Normally several complete paragraphs will fit into the

buffer space available; and it becomes quite reasonable to conduct the search in "scheduled order" on, say, a paragraph by paragraph basis, thus avoiding some unnecessary searching.

When the search context is too short, a potential new source of undesirable overhead appears, namely, that associated with setting up a large number of unsuccessful searches in short context units. In such cases, an initial "global" scan is employed, followed by a "local" search whenever a responding context is tentatively identified.

Because of explicit and implicit requirements for interaction between the search and print control routines, the choice of specific "local" and "global" contexts depends on what contexts are to be printed as well as what contexts are to be searched. Searches are divided into four types, as shown in Table 6.5. The first section of the table defines the search type, and the remainder lists the search contexts which are employed on three levels. Level 0 context delimiters define the total scope of the search within each document; Level 1 context delimiters define the context in which searching is normally conducted (the "global" context); Level 2 context delimiters define the range of the detailed searches (the "local" context) which are sometimes required.

6.3.1.1 Type I Search

The requested major context unit is searched by memory loads for all search terms until the search request is satisfied or the document is rejected. Whenever a responding document is detected, the PRINT routine is called, and all requested contexts in the document are printed. Searching is resumed with the next document in the data base.

6.3.1.2 Type II Search

Processing proceeds exactly as for a Type I search except that the

Table 6.5. Search Control Classifications

Definition		SEARCH TYPE I	SEARCH TYPE II	SEARCH TYPE III	SEARCH TYPE IV
	FIND Context	Major (C-Delimiters)	Bibliographic Subsection (e.g., TITLE) (D-Delimiters)	Minor (E-Delimiters)	Minor (E-Delimiters)
	PRINT Context	Any	Any	All Major or Bibliographic Subsections (C- or D-Delimiters)	Any Minor (E-Delimiters)
	Level 0 Context	FIND Context	FIND Context	Abstract - End	Abstract - End
	Level 1 Context	Memory Buffer	FIND Context	Memory Buffer	Paragraph
	Level 2 Context	---	---	Same as FIND Context (Paragraph or Sentence)	Same as FIND Context (Paragraph or Sentence)

"scheduled" search mode is employed since, by a design requirement, the entire Bibliographic DATA section is known to be in memory.

6.3.1.3 Type III Search

When SENTENCE or PARAGRAPH is specified as the FIND context, the search is necessarily restricted to those major context units which contain sentence and paragraph subdivisions: TEXT, REFERENCES and COMMENTS. Hence, Table 6.5 lists a Level 0 context extending from the beginning of the abstract to the end of the document. For Type III searches, the Level 1 context is again a full memory load, and the Level 2 context is the same as the FIND context.

A Level 1 search is conducted in "scheduled order" through the entire memory until any search term has been located. When this occurs, the Level 1 search is suspended; and a complete "scheduled" search is conducted in the responding Level 2 context (sentence or paragraph). If this Level 2 search is successful the PRINT routine is called, and printing proceeds as in Cases I and II. If the Level 2 search is not successful, the Level 1 search is resumed exactly where it left off.

6.3.1.4 Type IV Search

This case differs from Type III in that either sentence or paragraph has been selected as a PRINT context, and hence it will be necessary to print specific small context units. This is accomplished without a great deal of extra searching or bookkeeping by calling the output routines as soon as such a context can be identified. Control passes back and forth between the routines which control searching and those which control printing in such a way that a responding context unit is printed immediately, and searching is resumed in the next Level 1 or Level 2 context unit, as appropriate. This is the only case in

which searching continues in a given document after part of that document has been printed.

To facilitate this arrangement, the Level 0 context is Abstract-End, as before, but the Level 1 search is conducted on a paragraph by paragraph basis. If a paragraph is found to satisfy the search request, the search routine conducts a sentence by sentence search, if necessary, within that responding paragraph. When a search is completed successfully, the address of the responding context unit is passed to the print control routines which print the required context unit and return control, along with the address of the last character printed. Searching then resumes in the next sentence or paragraph, as appropriate. This processing is complicated somewhat by the fact that any of the four combinations of minor FIND and PRINT contexts is allowed: PARAGRAPH/PARAGRAPH, PARAGRAPH/SENTENCE, SENTENCE/PARAGRAPH or SENTENCE/SENTENCE.

6.3.2 Sentence and Paragraph Restrictions

A second problem arising from the non-uniform lengths of sentences and paragraphs is that it is very inefficient to guarantee that every memory load of text begins at the beginning of a paragraph and ends at the end of one. In fact, it is not considered practical to make this guarantee even for sentences. Hence, it is possible for a user to request a search for a character string which begins in one buffer and ends in the next, or to specify the co-occurrence of two terms which actually appear together in the required context but which do not lie in the same memory load. In such cases the search would fail, and the document would not be retrieved. Except as explained under "Type I Search", no attempt is made to continue a particular search from one memory buffer to another.

An effort has been made to load the disk in such a way as to minimize this problem. No word is ever divided between two sectors of the disk, and sentences are so divided only if it is impractical (in terms of wasted disk space) or impossible to do otherwise.

7.0 PRINT Control

Actual formatting and printing of document text is under the control of two routines, PRNT and BLKP, which interact with the search procedures as explained in Section 6.

Whenever the print control routine (PRNT) is called, it first checks certain flags to determine whether this is the first or last call or an intermediate call for the current document. An intermediate call indicates that a Type IV search is in progress and a responding minor context has just been located. In that case, the required paragraph or sentence is printed, and control is returned to the search procedure.

When a first or last call is received, the CTXT table is scanned to determine what context should be printed next, and a check is made to determine whether this context has already been printed. If it has, the CTXT table scan is resumed; if not, print processing proceeds. If the next context to be printed is SENTENCE or PARAGRAPH, processing begins at the start of the first major context section which contains these subdivisions and which has not already been printed.

Print processing consists of breaking the text first into paragraphs and then into individual print lines, determining whether or not each line contains any of the requested search terms and, if so, supplying the required marginal characters.

In order to identify those lines containing search terms, it is necessary to scan the entire text to be printed once for each term in the search request. (Thus, this facility represents a fairly large penalty in terms of processing time, program space and program complexity.) The search is conducted on a paragraph by paragraph basis*; and when a search term is found, a special

---

*If the PRINT context contains no paragraph subdivisions, then the search is conducted in the full PRINT context, if in core, or in the full current text buffer.

non-printable character, :88:, is placed in the first blank preceding the end of the search term. When all search terms have been marked in this way, the routine BLKP divides the paragraph into lines of maximum length 125 characters ending with blank, supplies the appropriate marginal prefix ('**---' if the line contains a search term or '-----' if it does not), and prints the line.

In order to conserve memory space and (hopefully) reduce overhead, print processing is conducted by paragraphs rather than by lines, and it takes place entirely within the original text buffer. Lines are not moved to a special location before printing.

After a paragraph has been printed, the procedure is repeated for the next paragraph or the CTXT table scan is resumed to locate the next context for printing. When all requested printing has been completed, control is returned to the search routines, and processing begins in the next document.

8.0 Implementation of Negation

The user does not presently have the capability of specifying that a term should not be present in a search context. However, this facility can be implemented easily as follows.

The second, third and fourth bits from the left of the tag word are not used for any purpose: one of these could be designated as the "negation bit". In constructing a new tag, then, the parsing procedure would first set this bit to zero, and then reverse its state for each occurrence of the "NOT" operator before the entity to which the tag is assigned. The remainder of the tag would be constructed as before.

The rest of the scheduling and searching procedures could work exactly as before except that the SUCCESS and FAILURE pointers would be interchanged for any entity which was negated.

Including negation might make it desirable to change the scheduling procedures. That possibility requires further investigation.

9.0 System Errors

There are six locations in the SRCH and PRNT routines from which errors in programming or data base formatting can cause the message "ERR--SYSTEM", to be printed. These locations are listed below together with a description of the condition which causes the failure. The message itself does not, at present, indicate which error has occurred.

STATEMENT LABEL	CAUSE OF FAILURE
SRCH11	Failure to find the Level 0 start character after the sector in which it occurs has just been read from disk or verified as present in core.
SRCH21	Failure to find a paragraph symbol (:EC:) marking the start of the next Level 1 search context (Type IV search). A paragraph symbol should always be detected, even if it follows the end-of-buffer character (:88:).
PRNT11	Failure to find a context start symbol for searching by PRNT.
PRNT15AA	Failure to find a paragraph symbol which should occur within a larger context for searching by PRNT.
PRNT17D	Improper address calculation in preparation to read disk sector containing start of BODY.
PRNT17M	Failure to find start of BODY character after verifying its presence in core.

10.0 S-Level Debugging Facilities: the DUMP and CHANGE Instructions

Two special instructions, DUMP and CHANGE, have been included in the initial implementation of the retrieval program to assist in debugging. These instructions are intended for use by system maintenance personnel, and in the interest of economy of storage and programming effort, very rigid input formats must be observed. Both of these instructions should be omitted from "production" versions of the program since both can cause alteration of the S-Memory.

10.1 DUMP Instruction

The DUMP instruction causes the printing of a hexadecimal dump of a selected portion of the S-Memory. The output contains 16 words per line, with the address of the first word on each line divisible by 16.

The input format for the DUMP instruction is

DUMP_XXXXZZZZ

where "DUMP" must be the first four characters on the input line and must be followed by exactly one blank. "X" and "Z" in this definition represent hexadecimal characters (0-9, A-F), and the string "XXXXZZZZ" contains the actual object code for the desired instruction (see appropriate microcode documentation).

10.2 CHANGE Instruction

The CHANGE instruction allows the user to change the contents of selected words in S-Memory. Its format is

CHANGE_XXXX_Y__ZZZZ....ZZ

where X, Y and Z are hexadecimal characters. Again, the characters "CHANGE" must be the first five characters on the input line, and blanks must appear exactly as shown.

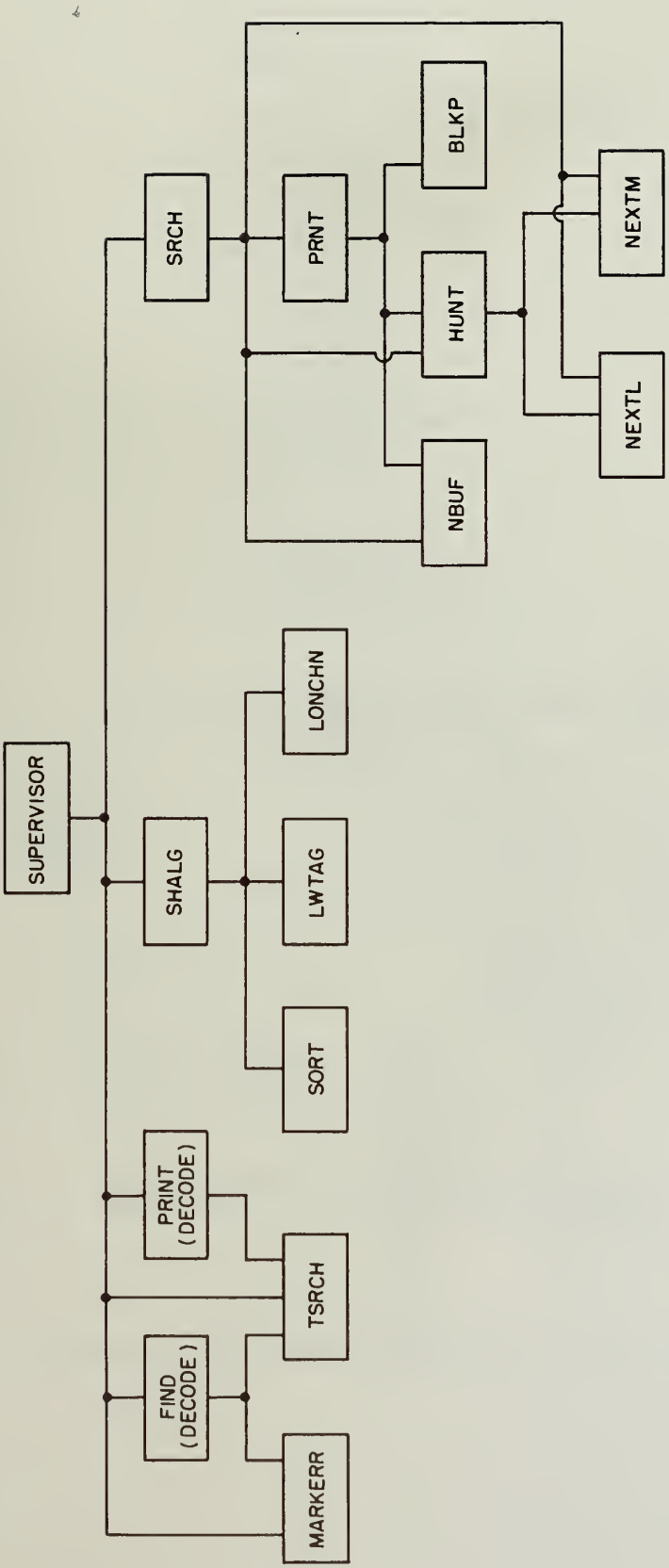
"XXXX"	is the address of the first word to be changed.
"Y"	is the number of consecutive words to be changed (0-9).
"ZZZZ....ZZ"	is the hexadecimal character string to be loaded at "XXXX", four characters per 16-bit word.

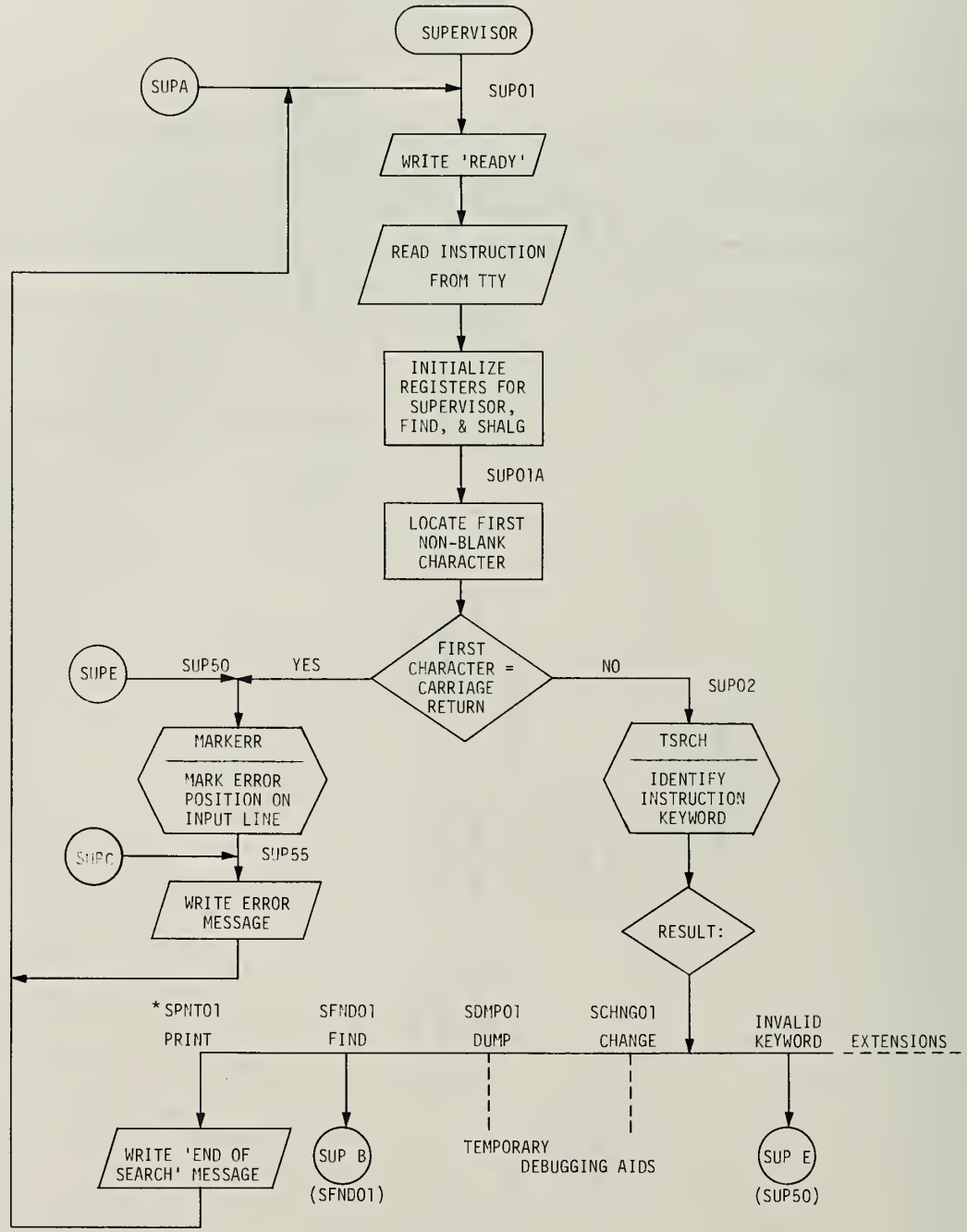
APPENDIX

APPENDIX A

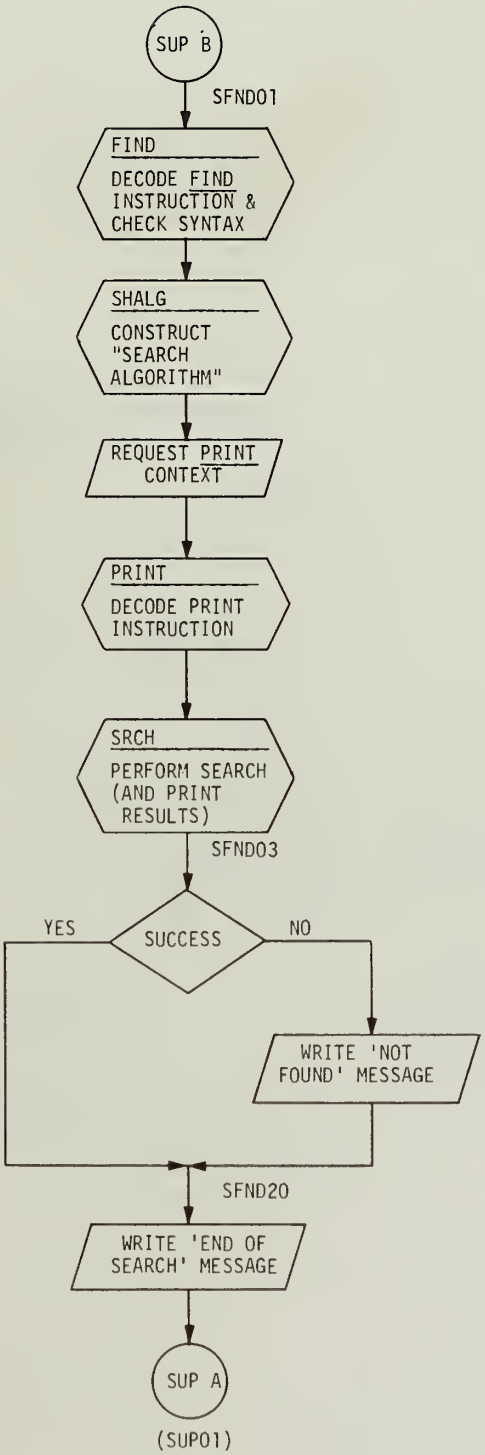
Flow Charts

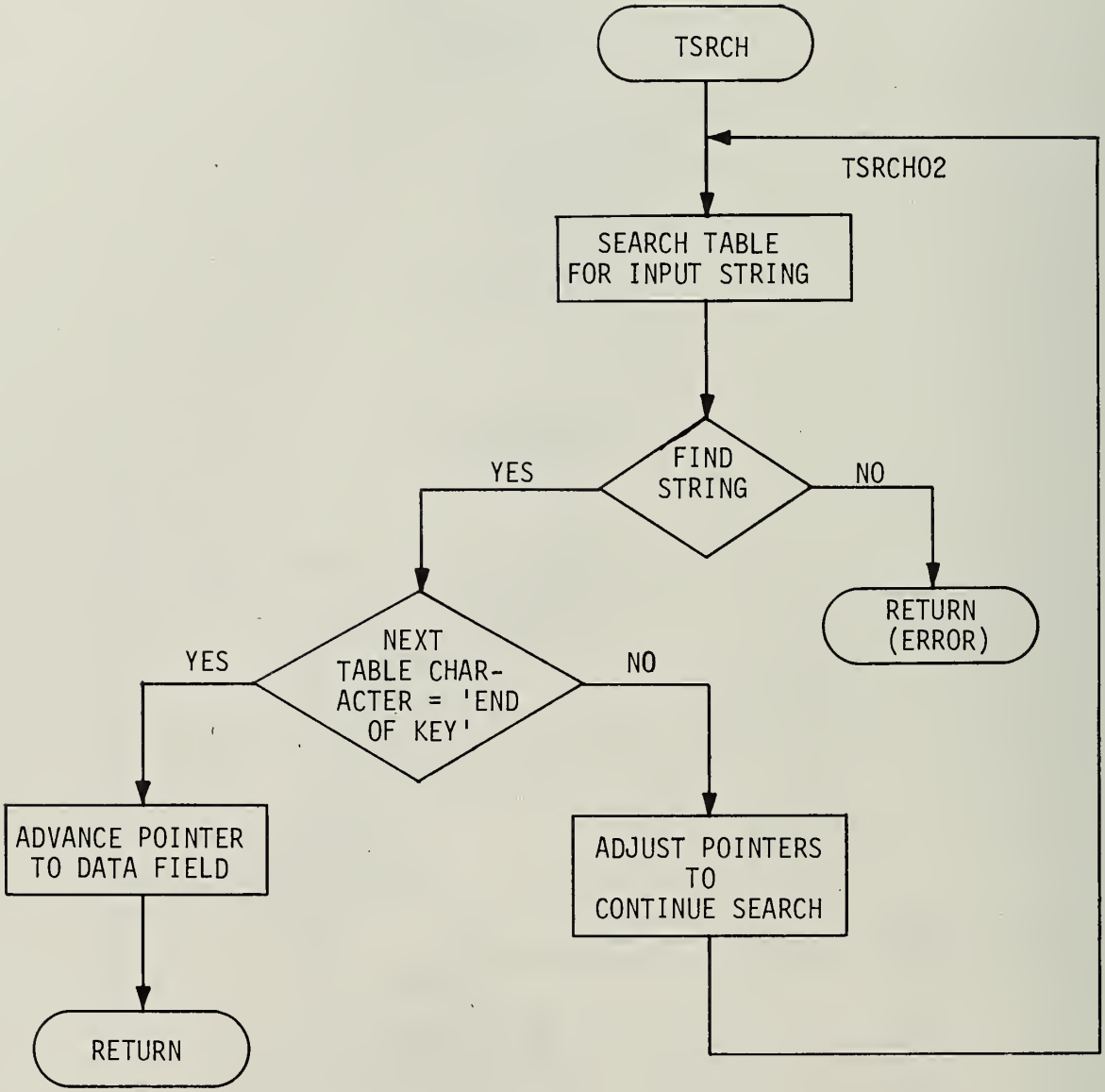
This section contains detailed flow charts for the sixteen processing routines that make up the retrieval control program. The first diagram shows the overall structure of the program and the interrelationships which exist among the parts. When one routine is shown below another and connected to it by a vertical line, the first procedure is called by the second. The remainder of the section contains standard flow diagrams.

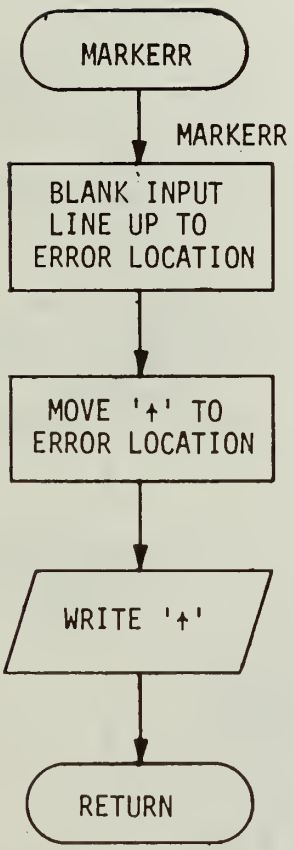

SUPERVISOR
SHALG
LONCHN
LWTAG
SORT
PRINT (DECODE)
TSRCH
FIND (DECODE)
MARKERR
SRCH
PRNT
BLKP
HUNT
NEXTM
NEXTL
NBUF

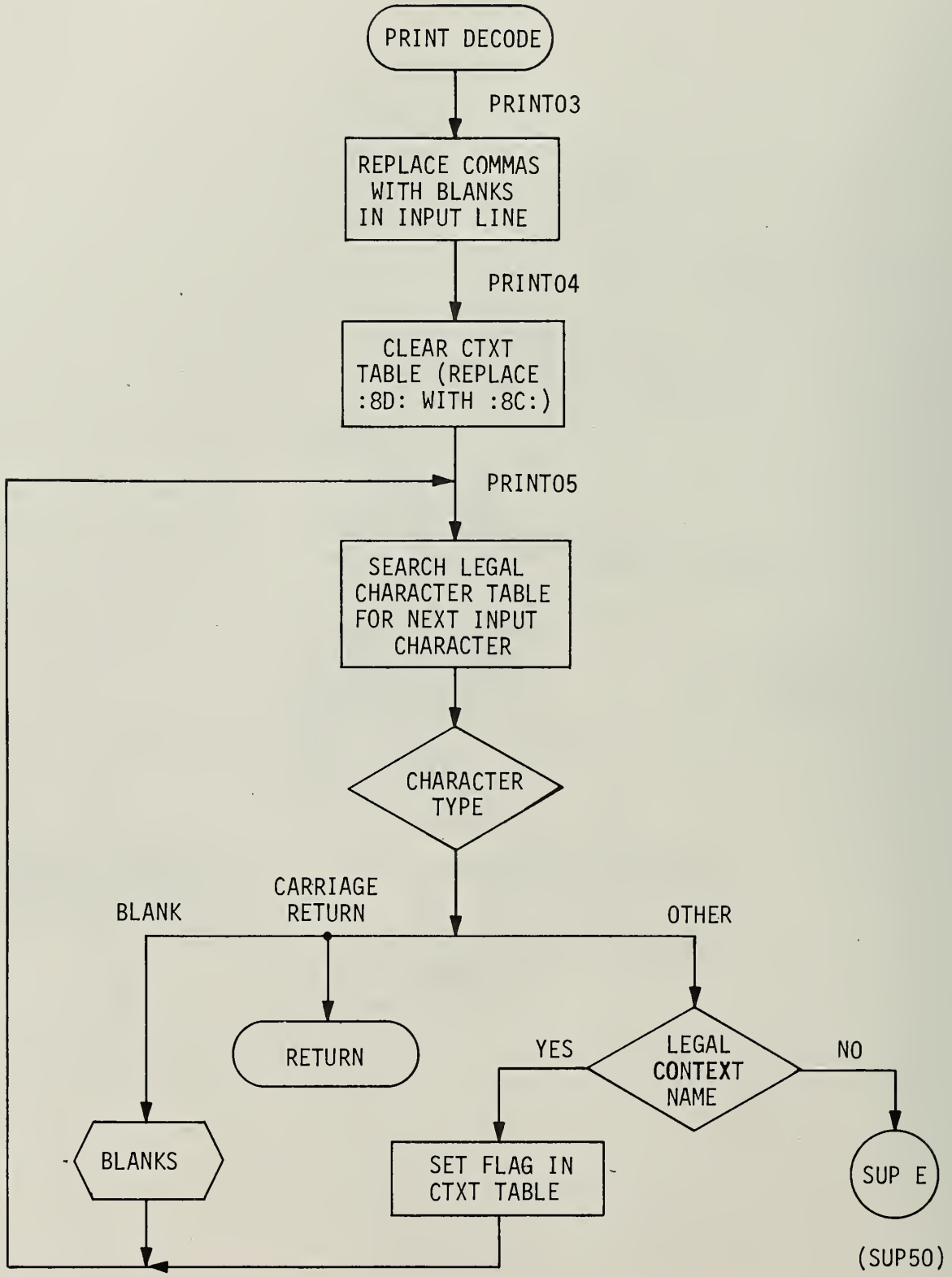


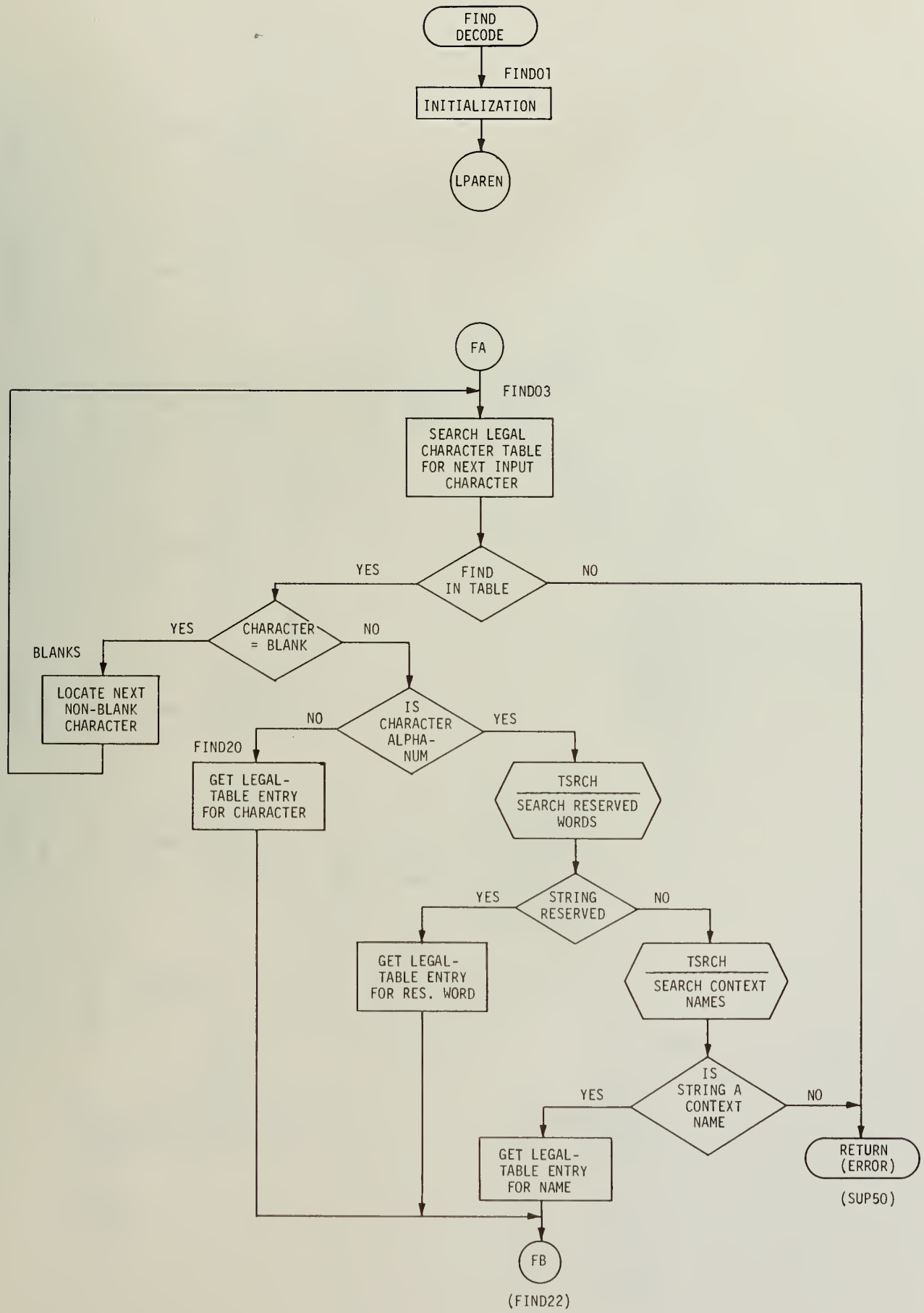
*THE LABEL "SPNT01" IS AVAILABLE FOR FUTURE IMPLEMENTATION OF AN INDEPENDENT "PRINT" INSTRUCTION.

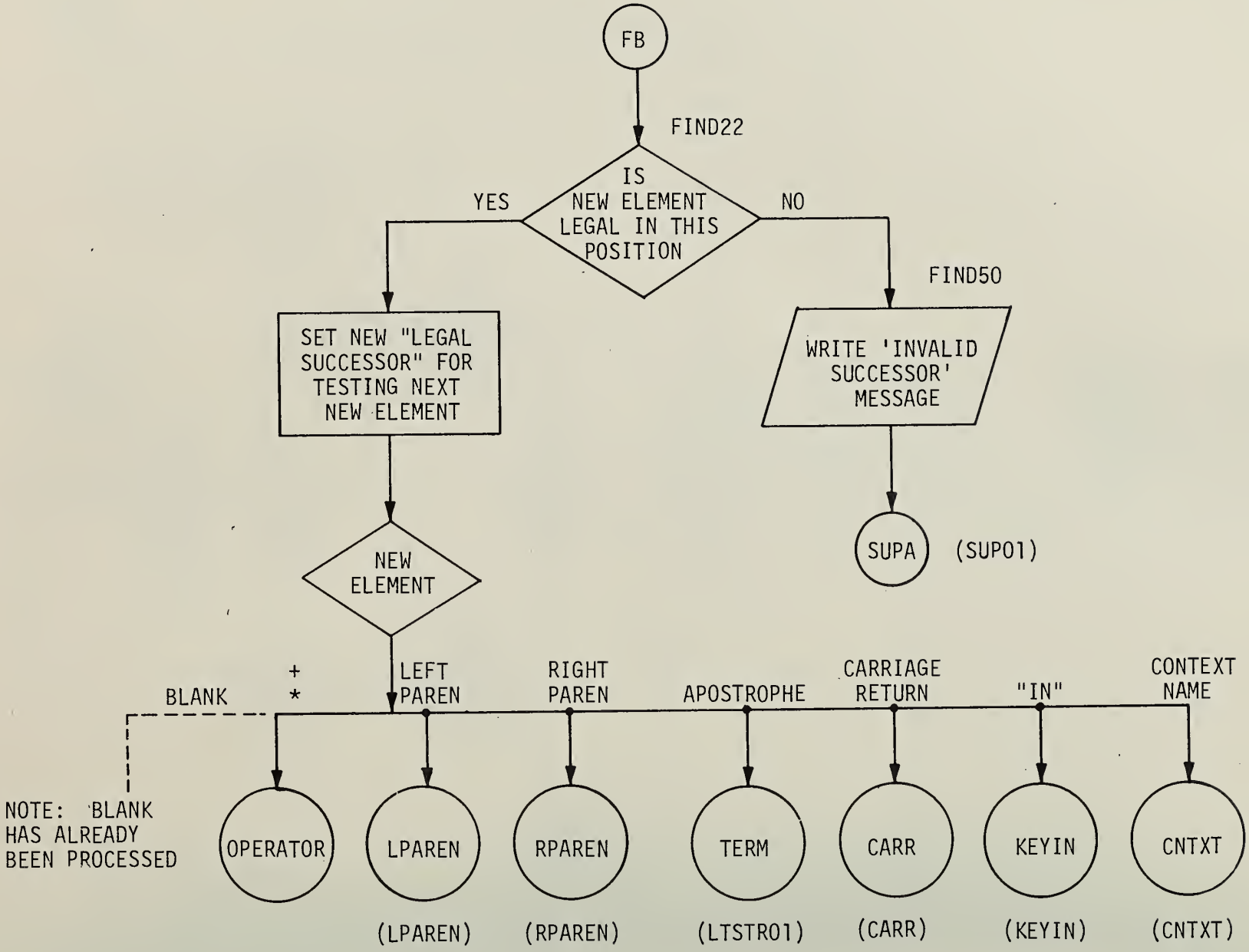

SUP B
SFND01
FIND
DECODE FIND INSTRUCTION & CHECK SYNTAX
SHALG
CONSTRUCT "SEARCH ALGORITHM"
REQUEST PRINT CONTEXT
PRINT
DECODE PRINT INSTRUCTION
SRCH
PERFORM SEARCH (AND PRINT RESULTS)
SFND03
YES
SUCCESS
NO
WRITE 'NOT FOUND' MESSAGE
SFND20
WRITE 'END OF SEARCH' MESSAGE
SUP A
(SUP01)

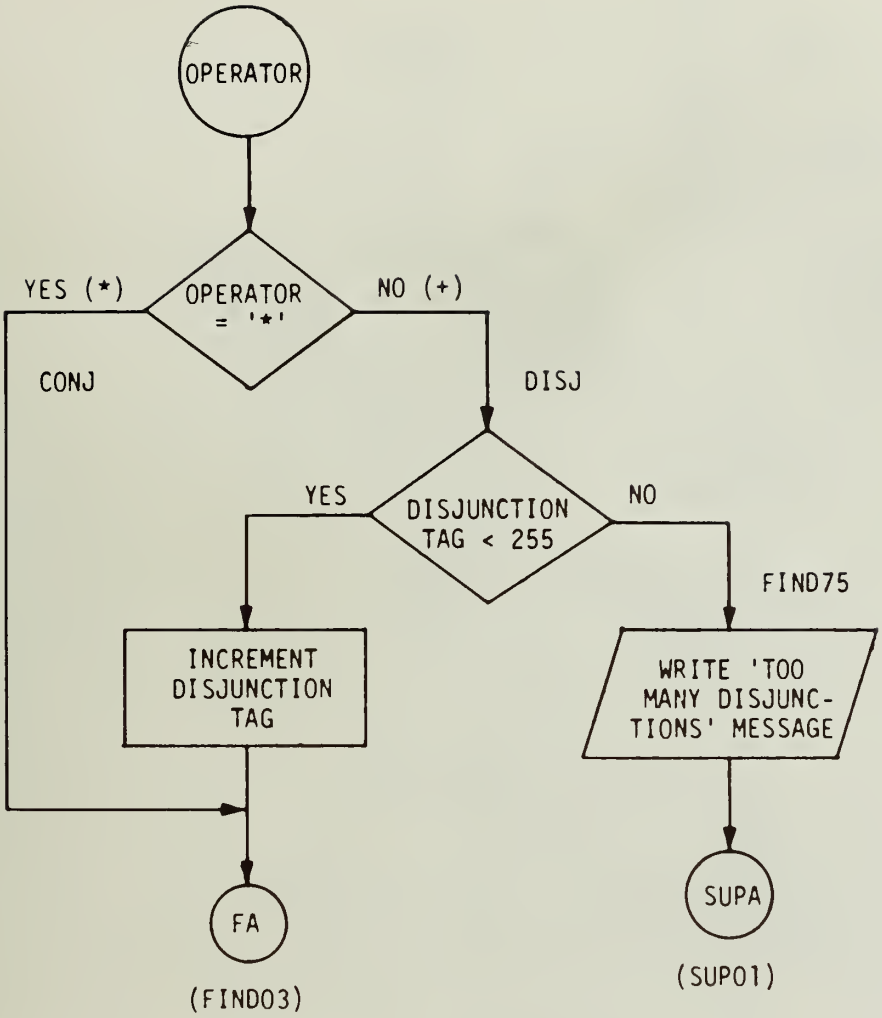

TSRCH
TSRCH02
SEARCH TABLE FOR INPUT STRING
FIND STRING
YES
NO
RETURN (ERROR)
NEXT TABLE CHAR-ACTER = 'END OF KEY'
YES
NO
ADVANCE POINTER TO DATA FIELD
ADJUST POINTERS TO CONTINUE SEARCH
RETURN

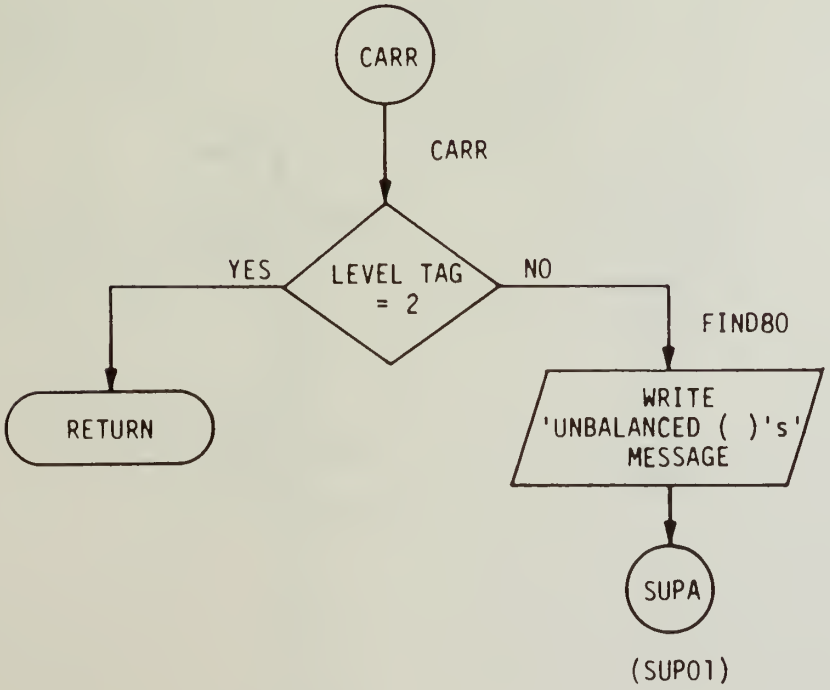

MARKERR
MARKERR
BLANK INPUT LINE UP TO ERROR LOCATION
MOVE '↑' TO ERROR LOCATION
WRITE '↑'
RETURN

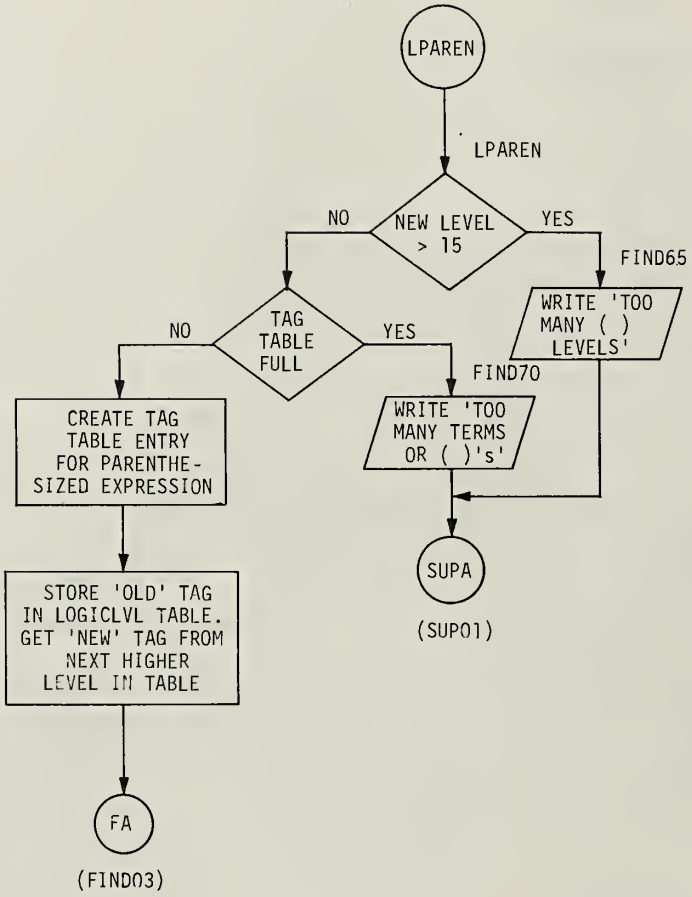

PRINT DECODE
PRINT03
REPLACE COMMAS WITH BLANKS IN INPUT LINE
PRINT04
CLEAR CTXT TABLE (REPLACE :8D: WITH :8C:)
PRINT05
SEARCH LEGAL CHARACTER TABLE FOR NEXT INPUT CHARACTER
CHARACTER TYPE
BLANK
CARRIAGE RETURN
OTHER
RETURN
BLANKS
YES
LEGAL CONTEXT NAME
NO
SET FLAG IN CTXT TABLE
SUP E
(SUP50)

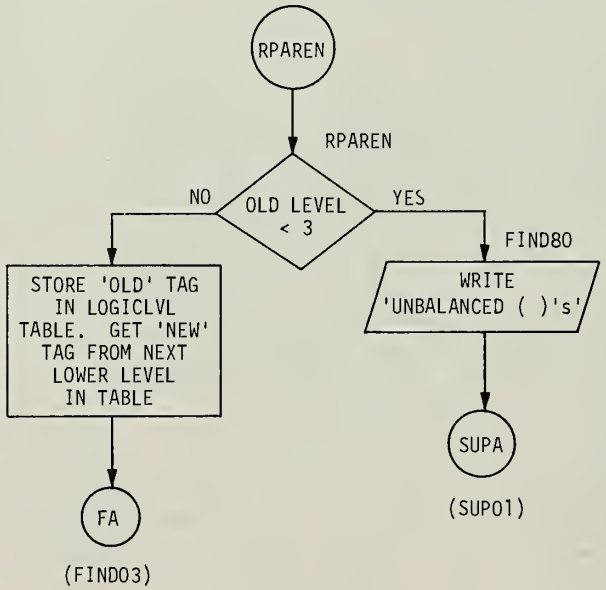

FIND DECODE
FIND01
INITIALIZATION
LPAREN
FA
FIND03
SEARCH LEGAL CHARACTER TABLE FOR NEXT INPUT CHARACTER
FIND IN TABLE
YES
NO
CHARACTER = BLANK
YES
NO
BLANKS
LOCATE NEXT NON-BLANK CHARACTER
IS CHARACTER ALPHA-NUM
NO
YES
FIND20
GET LEGAL-TABLE ENTRY FOR CHARACTER
TSRCH
SEARCH RESERVED WORDS
STRING RESERVED
YES
NO
GET LEGAL-TABLE ENTRY FOR RES. WORD
TSRCH
SEARCH CONTEXT NAMES
IS STRING A CONTEXT NAME
YES
NO
GET LEGAL-TABLE ENTRY FOR NAME
RETURN (ERROR)
(SUP50)
FB
(FIND22)

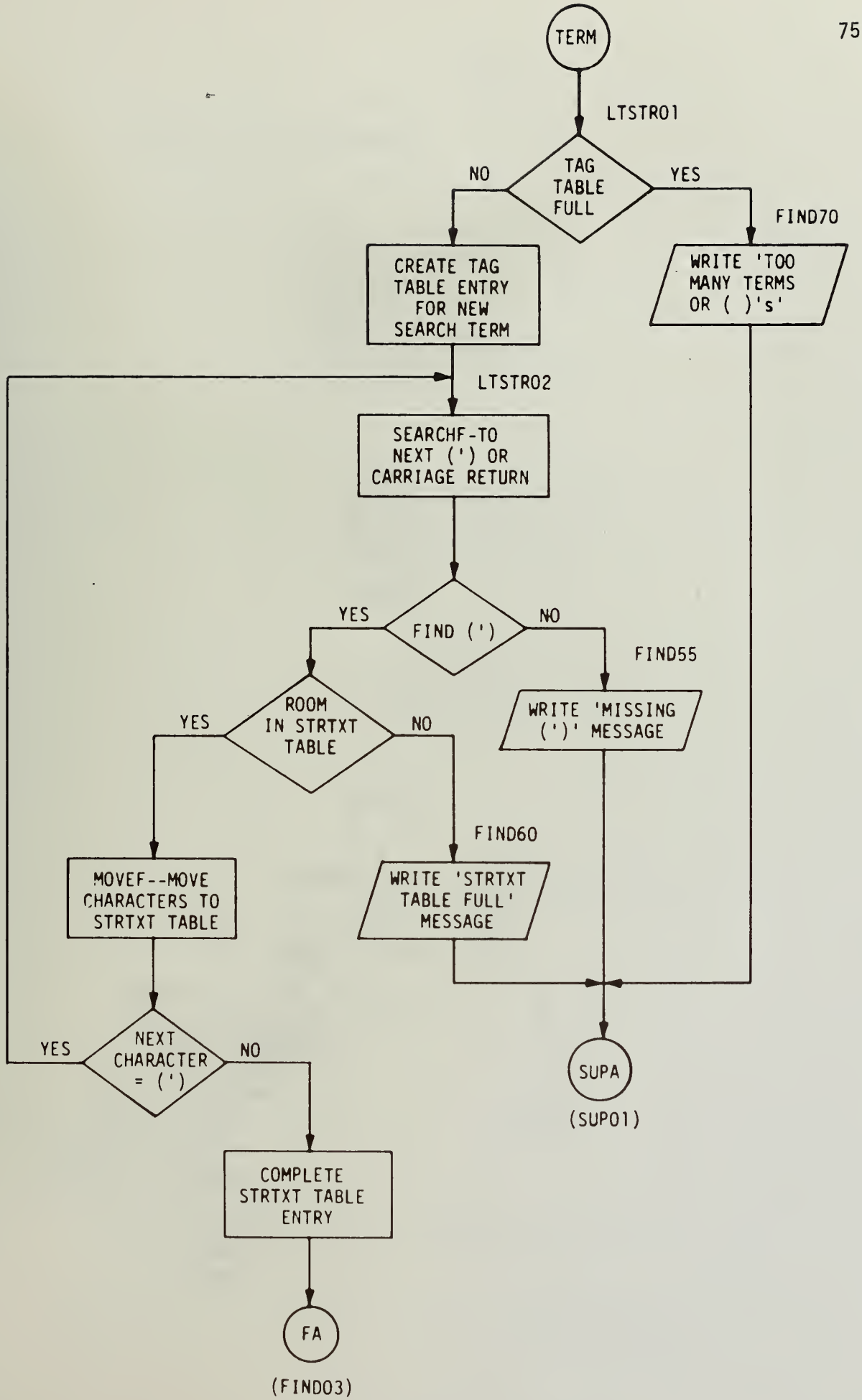

FB
FIND22
IS NEW ELEMENT LEGAL IN THIS POSITION
YES
NO
FIND50
SET NEW "LEGAL SUCCESSOR" FOR TESTING NEXT NEW ELEMENT
WRITE 'INVALID SUCCESSOR' MESSAGE
SUPA
(SUP01)
NEW ELEMENT
BLANK
+
*
LEFT PAREN
RIGHT PAREN
APOSTROPHE
CARRIAGE RETURN
"IN"
CONTEXT NAME
NOTE: BLANK HAS ALREADY BEEN PROCESSED
OPERATOR
LPAREN
RPAREN
TERM
CARR
KEYIN
CNTXT
(LPAREN)
(RPAREN)
(LTSTR01)
(CARR)
(KEYIN)
(CNTXT)

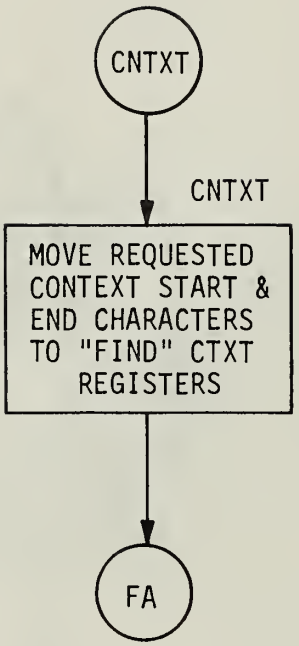

OPERATOR
OPERATOR = '*'
YES (*)
NO (+)
CONJ
DISJ
DISJUNCTION TAG < 255
YES
NO
FIND75
INCREMENT DISJUNCTION TAG
WRITE 'TOO MANY DISJUNC-TIONS' MESSAGE
FA
SUPA
(FIND03)
(SUP01)

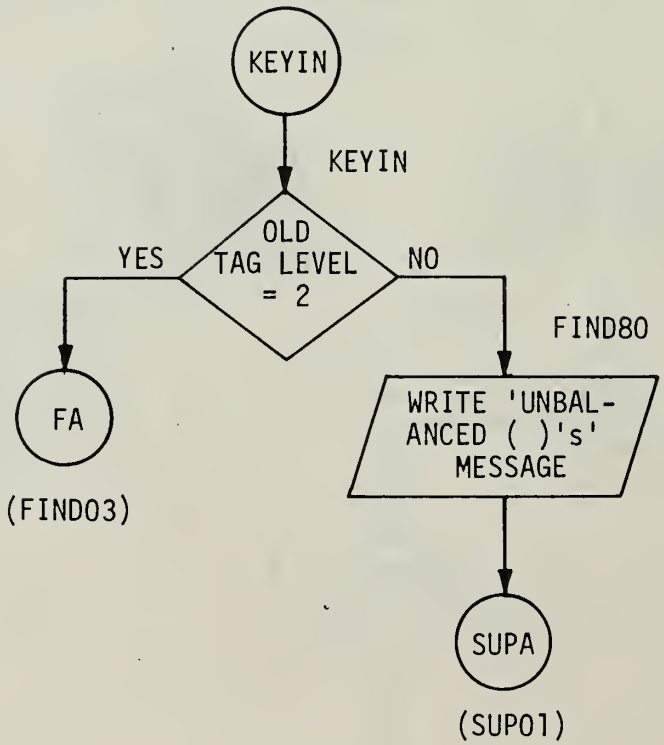

CARR
CARR
LEVEL TAG = 2
YES
NO
FIND80
RETURN
WRITE 'UNBALANCED ( )'s' MESSAGE
SUPA
(SUP01)

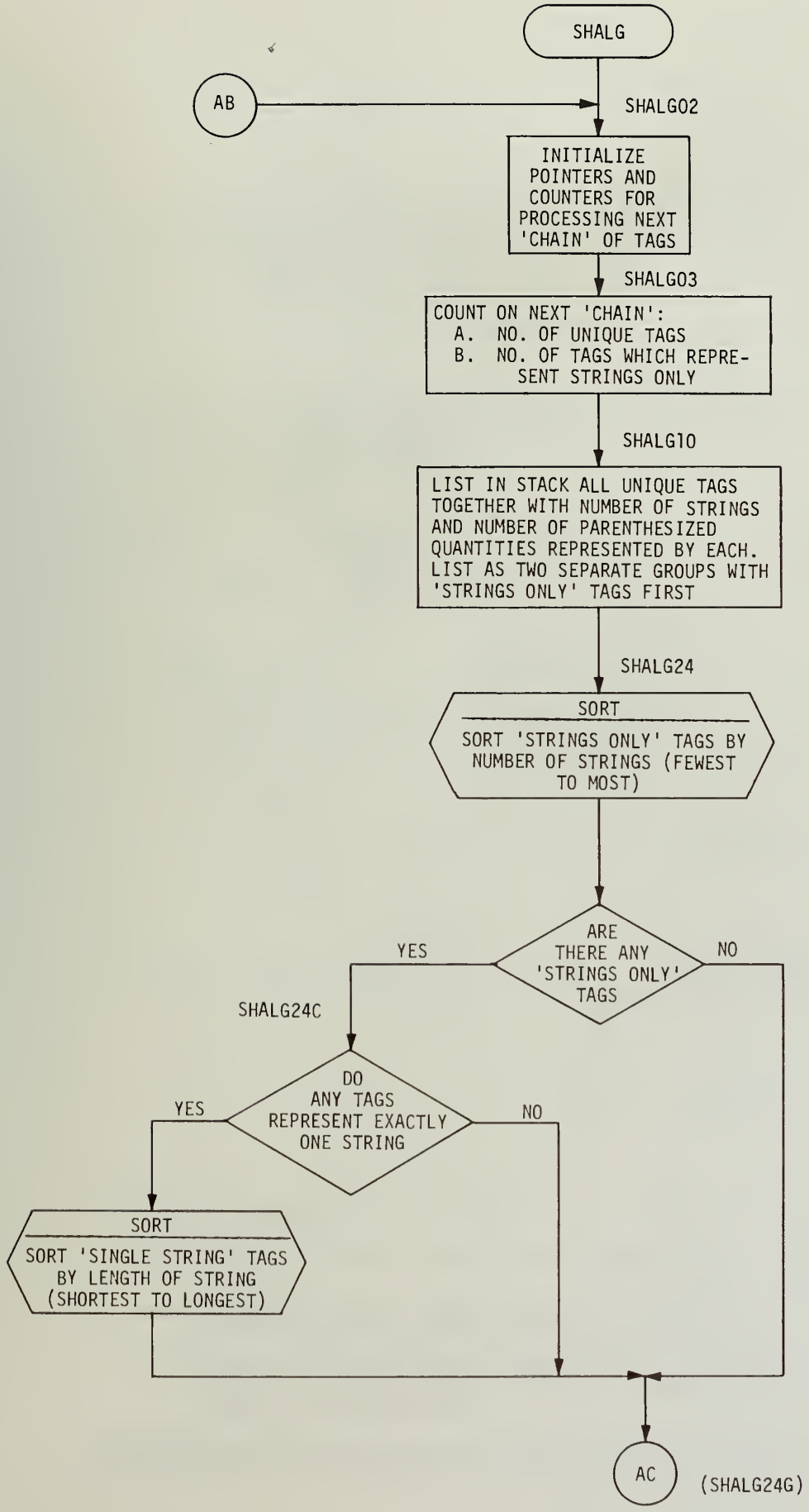

LPAREN
LPAREN
NEW LEVEL > 15
NO
YES
FIND65
WRITE 'TOO MANY ( ) LEVELS'
TAG TABLE FULL
NO
YES
FIND70
WRITE 'TOO MANY TERMS OR ( )'s'
CREATE TAG TABLE ENTRY FOR PARENTHE-SIZED EXPRESSION
SUPA
(SUP01)
STORE 'OLD' TAG IN LOGICLVL TABLE. GET 'NEW' TAG FROM NEXT HIGHER LEVEL IN TABLE
FA
(FIND03)

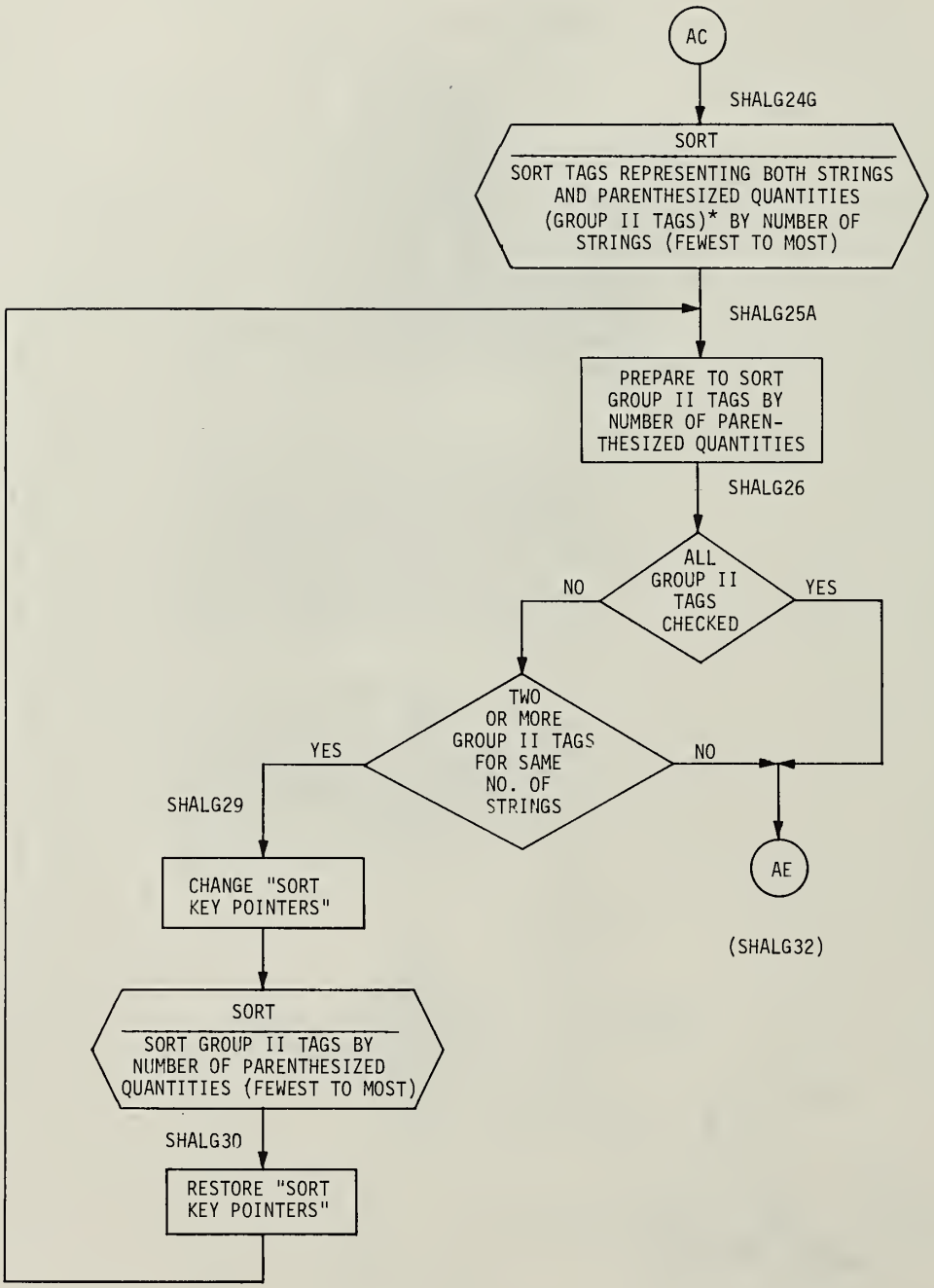

RPAREN
RPAREN
OLD LEVEL < 3
NO
YES
FIND80
WRITE 'UNBALANCED ( )'s'
STORE 'OLD' TAG IN LOGICLVL TABLE. GET 'NEW' TAG FROM NEXT LOWER LEVEL IN TABLE
SUPA
(SUP01)
FA
(FIND03)


TERM
LTSTR01
TAG TABLE FULL
NO
YES
FIND70
WRITE 'TOO MANY TERMS OR ( )'s'
CREATE TAG TABLE ENTRY FOR NEW SEARCH TERM
LTSTR02
SEARCHF-TO NEXT (') OR CARRIAGE RETURN
FIND (')
YES
NO
FIND55
WRITE 'MISSING (')' MESSAGE
ROOM IN STRTXT TABLE
YES
NO
FIND60
WRITE 'STRTXT TABLE FULL' MESSAGE
MOVEF--MOVE CHARACTERS TO STRTXT TABLE
NEXT CHARACTER = (')
YES
NO
SUPA
(SUP01)
COMPLETE STRTXT TABLE ENTRY
FA
(FIND03)

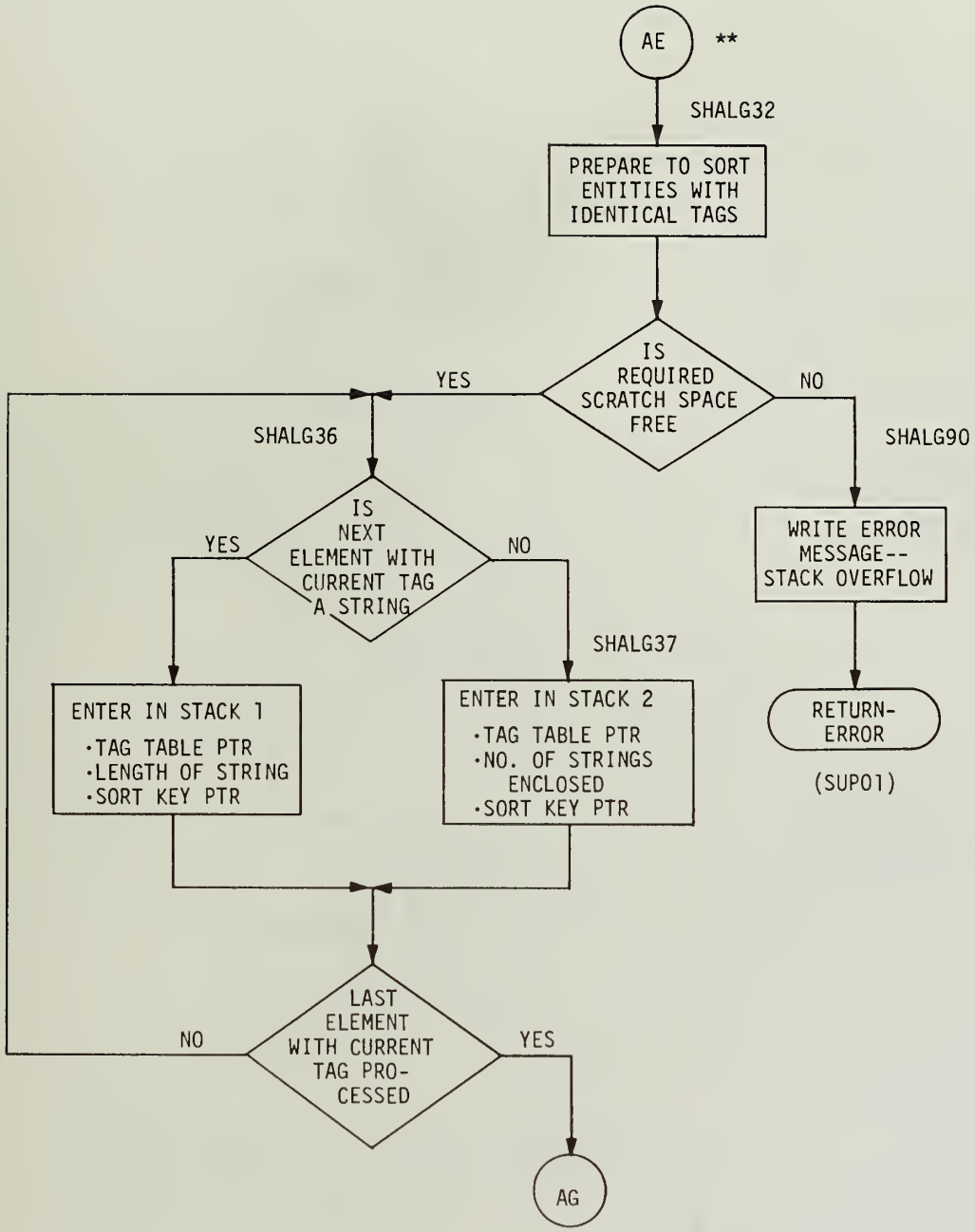

CNTXT
CNTXT
MOVE REQUESTED CONTEXT START & END CHARACTERS TO "FIND" CTXT REGISTERS
FA
(FIND03)


KEYIN
KEYIN
OLD TAG LEVEL = 2
YES
NO
FIND80
FA
(FIND03)
WRITE 'UNBAL-ANCED ( )'s' MESSAGE
SUPA
(SUP01)


SHALG
AB
SHALG02
INITIALIZE POINTERS AND COUNTERS FOR PROCESSING NEXT 'CHAIN' OF TAGS
SHALG03
COUNT ON NEXT 'CHAIN':
A. NO. OF UNIQUE TAGS
B. NO. OF TAGS WHICH REPRESENT STRINGS ONLY
SHALG10
LIST IN STACK ALL UNIQUE TAGS TOGETHER WITH NUMBER OF STRINGS AND NUMBER OF PARENTHESIZED QUANTITIES REPRESENTED BY EACH. LIST AS TWO SEPARATE GROUPS WITH 'STRINGS ONLY' TAGS FIRST
SHALG24
SORT
SORT 'STRINGS ONLY' TAGS BY NUMBER OF STRINGS (FEWEST TO MOST)
ARE THERE ANY 'STRINGS ONLY' TAGS
YES
NO
SHALG24C
DO ANY TAGS REPRESENT EXACTLY ONE STRING
YES
NO
SORT
SORT 'SINGLE STRING' TAGS BY LENGTH OF STRING (SHORTEST TO LONGEST)
AC
(SHALG24G)



*NOTE DEFINITION OF "GROUP II TAGS".



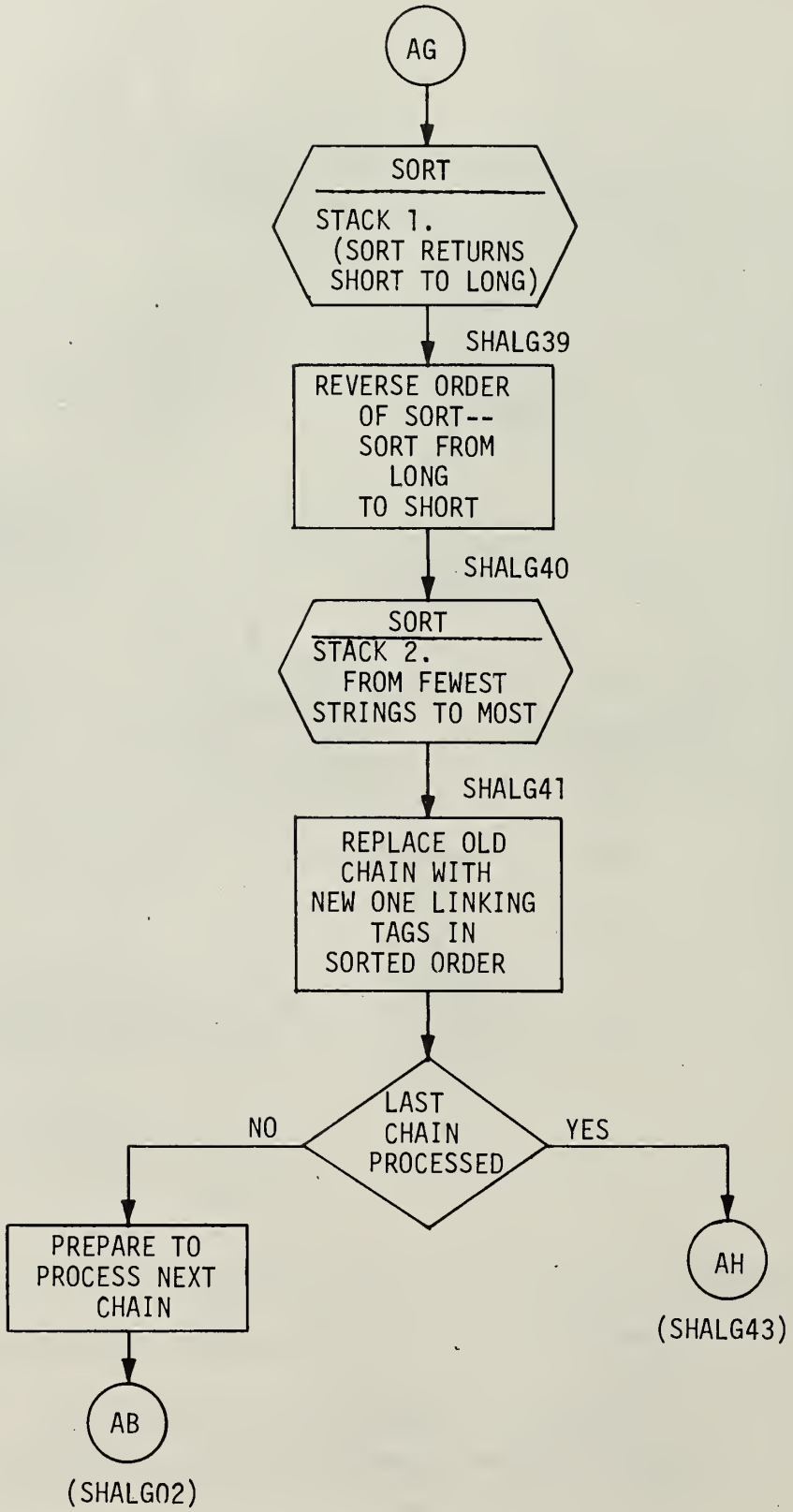
** AT THIS POINT, TAGS ON THE CHAIN CURRENTLY BEING PROCESSED HAVE BEEN SORTED INTO THE FOLLOWING ORDER:

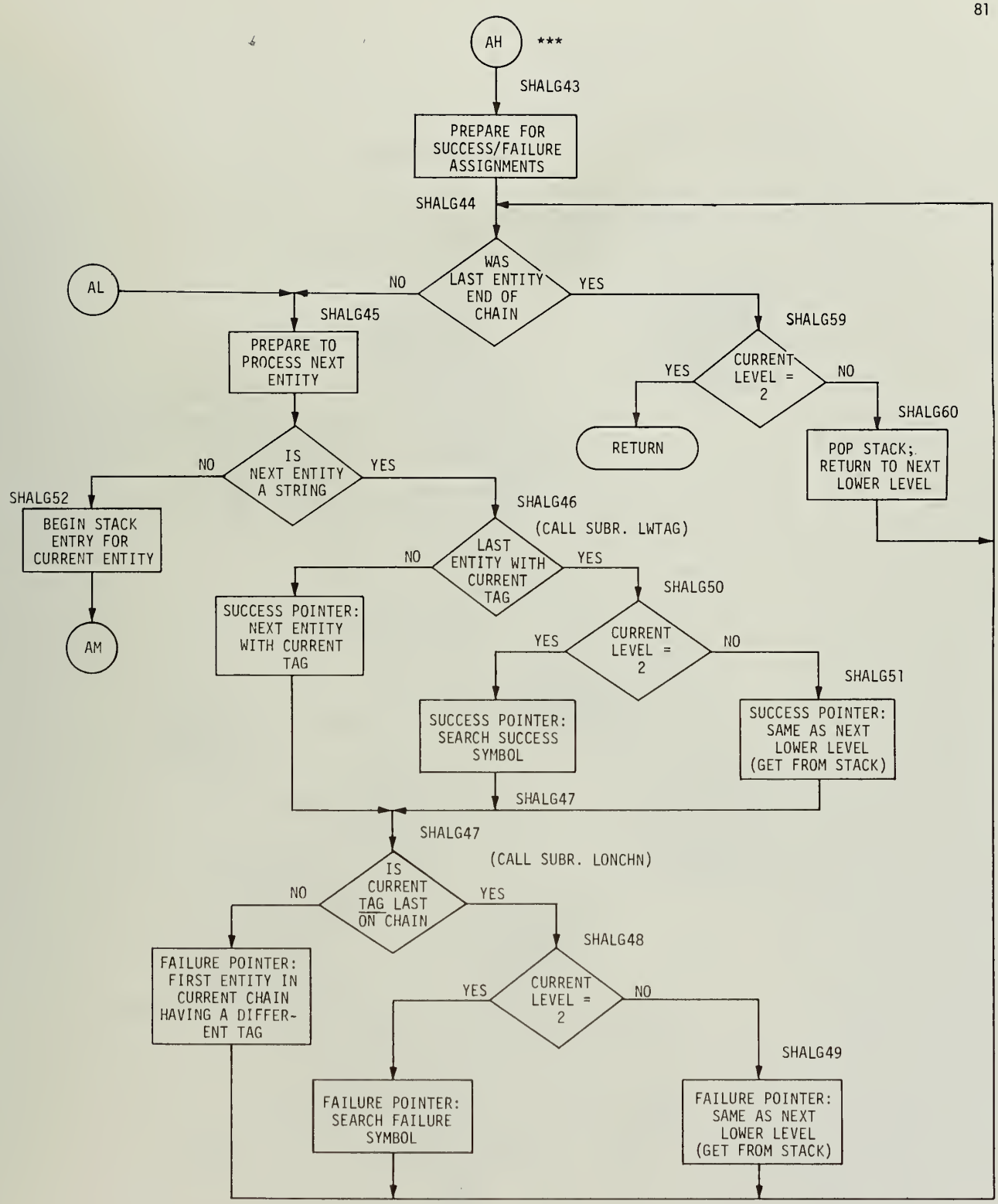
A. TAGS WHICH REPRESENT A SINGLE STRING, ARRANGED FROM SHORTEST STRING TO LONGEST

B. TAGS WHICH REPRESENT SEVERAL STRINGS (IN CONJUNCTION) BUT NO PARENTHESIZED EXPRESSIONS, ARRANGED FROM FEWEST STRINGS TO MOST

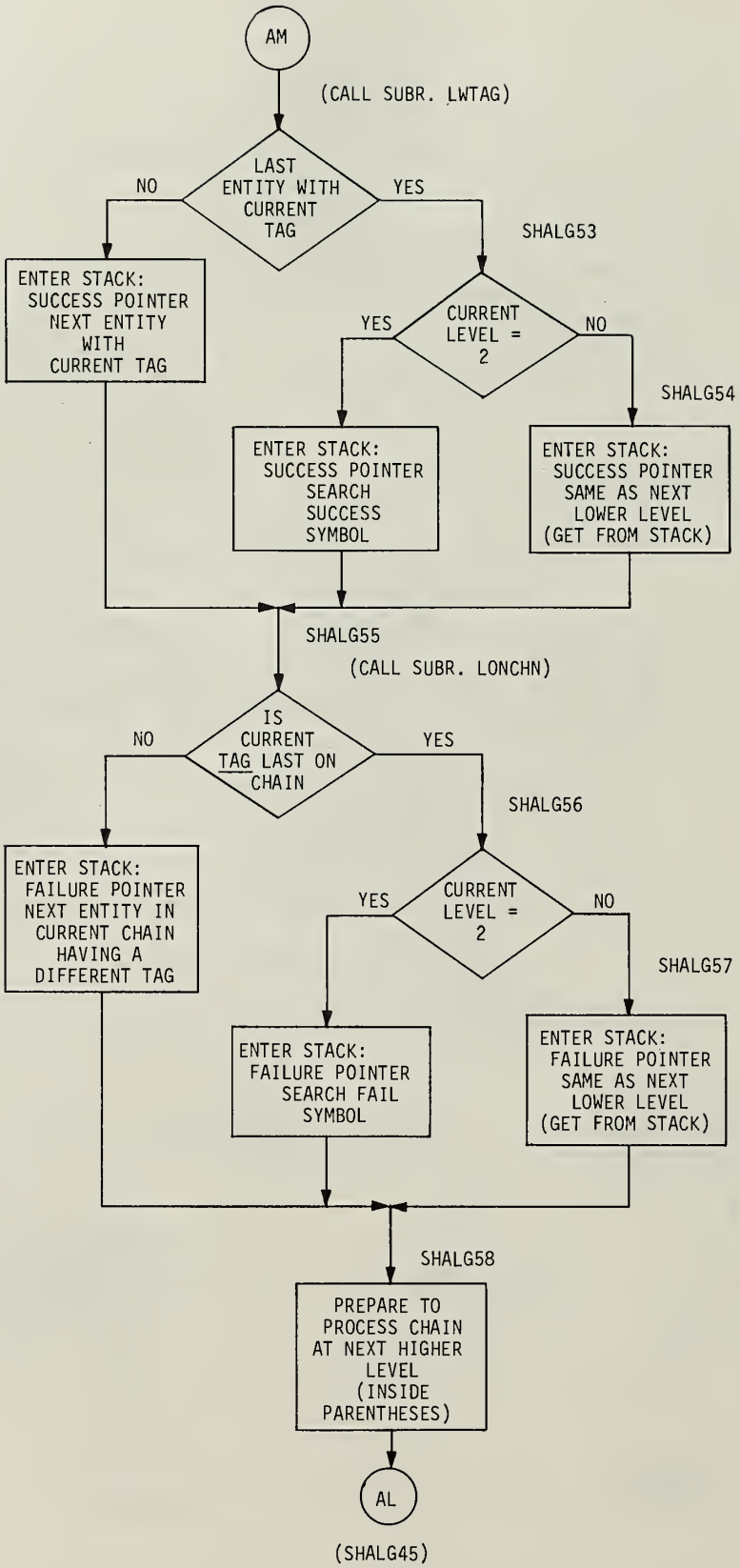
C. TAGS WHICH REPRESENT PARENTHESIZED QUANTITIES AND (POSSIBLY) STRINGS, ARRANGED FIRST FROM FEWEST STRINGS TO MOST AND THEN FROM FEWEST PARENTHESIZED EXPRESSIONS TO MOST

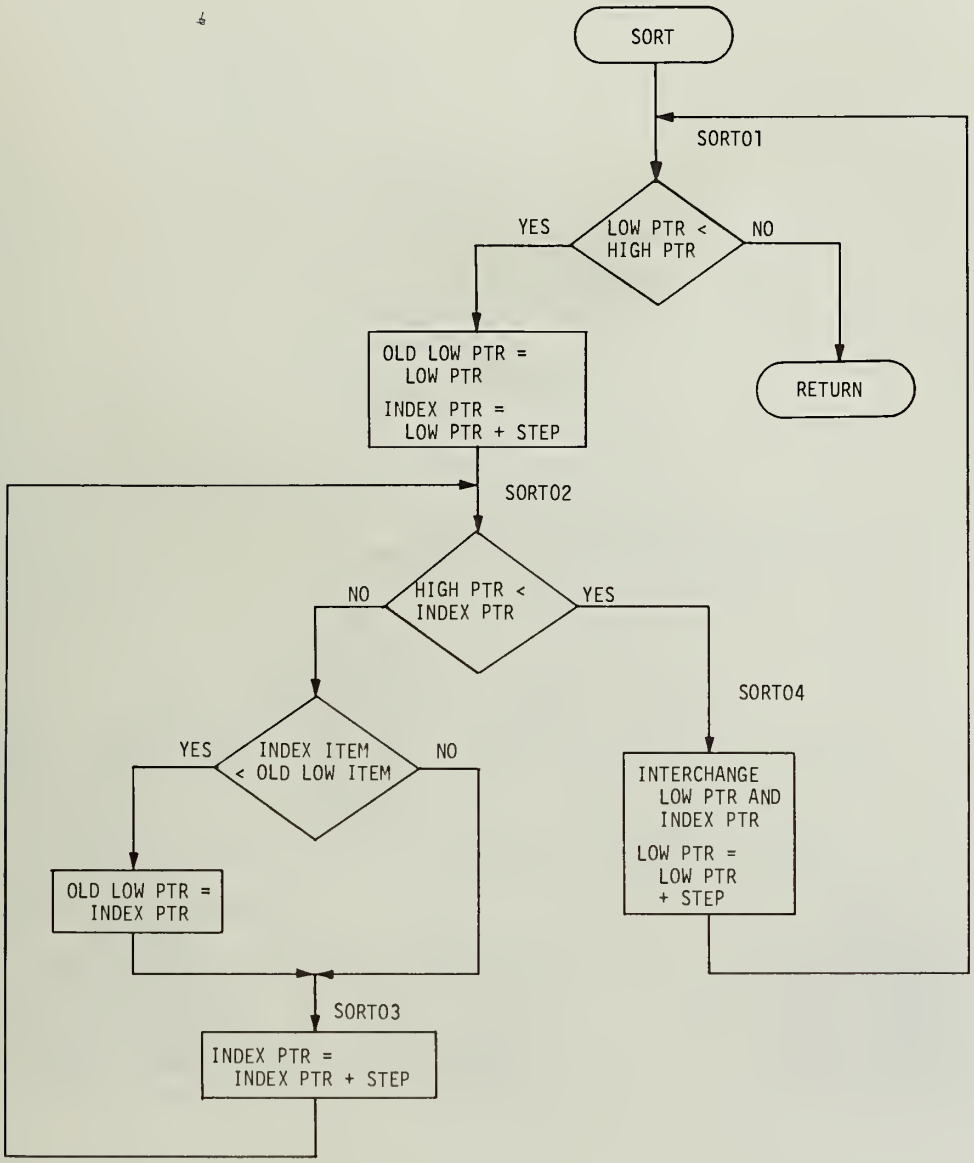
CODE WHICH FOLLOWS WILL, FOR EACH TAG, ARRANGE STRINGS FROM LONGEST TO SHORTEST AND PARENTHESIZED EXPRESSIONS FROM FEWEST ENCLOSED STRINGS TO MOST


AG
SORT
STACK 1.
(SORT RETURNS
SHORT TO LONG)
SHALG39
REVERSE ORDER
OF SORT--
SORT FROM
LONG
TO SHORT
SHALG40
SORT
STACK 2.
FROM FEWEST
STRINGS TO MOST
SHALG41
REPLACE OLD
CHAIN WITH
NEW ONE LINKING
TAGS IN
SORTED ORDER
LAST
CHAIN
PROCESSED
NO
YES
PREPARE TO
PROCESS NEXT
CHAIN
AB
(SHALG02)
AH
(SHALG43)



***MAKE SUCCESS/FAILURE ASSIGNMENTS.

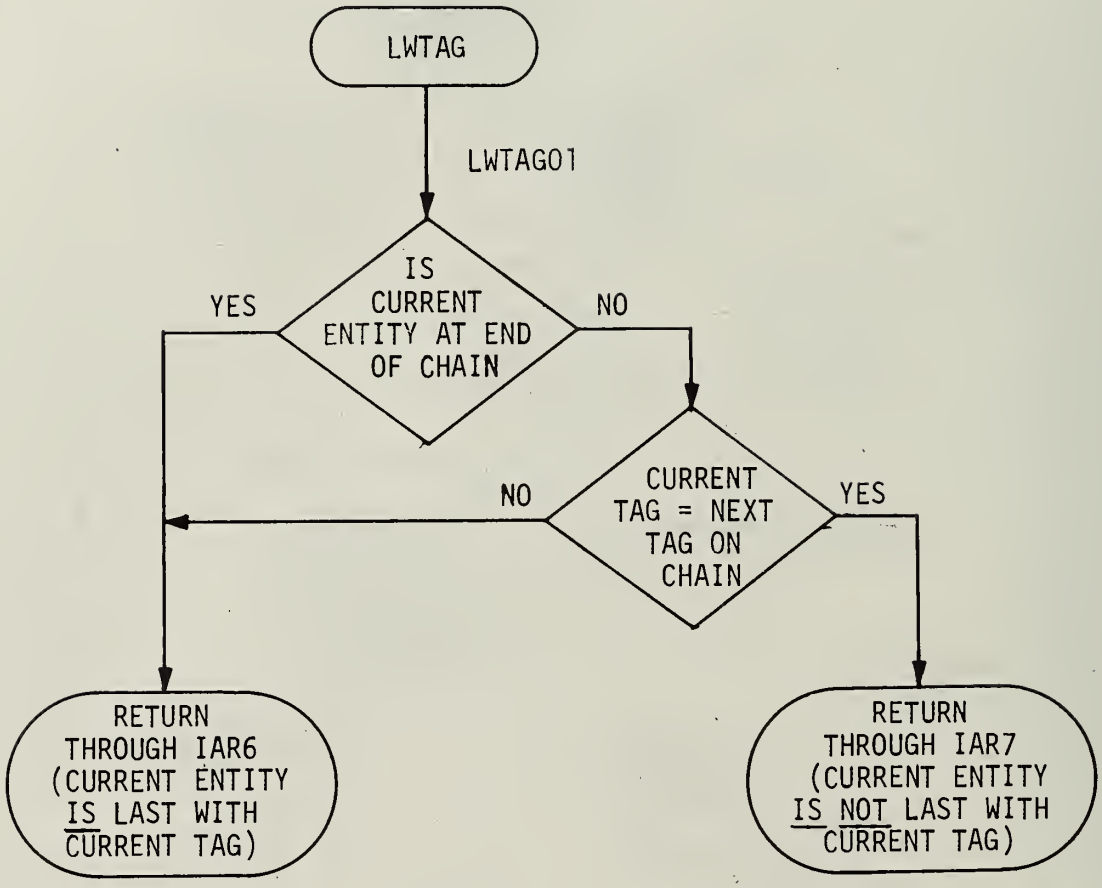

AM
(CALL SUBR. LWTAG)
LAST ENTITY WITH CURRENT TAG
NO
YES
ENTER STACK: SUCCESS POINTER NEXT ENTITY WITH CURRENT TAG
SHALG53
CURRENT LEVEL = 2
YES
NO
SHALG54
ENTER STACK: SUCCESS POINTER SEARCH SUCCESS SYMBOL
ENTER STACK: SUCCESS POINTER SAME AS NEXT LOWER LEVEL (GET FROM STACK)
SHALG55
(CALL SUBR. LONCHN)
IS CURRENT TAG LAST ON CHAIN
NO
YES
SHALG56
ENTER STACK: FAILURE POINTER NEXT ENTITY IN CURRENT CHAIN HAVING A DIFFERENT TAG
CURRENT LEVEL = 2
YES
NO
SHALG57
ENTER STACK: FAILURE POINTER SEARCH FAIL SYMBOL
ENTER STACK: FAILURE POINTER SAME AS NEXT LOWER LEVEL (GET FROM STACK)
SHALG58
PREPARE TO PROCESS CHAIN AT NEXT HIGHER LEVEL (INSIDE PARENTHESES)
AL
(SHALG45)

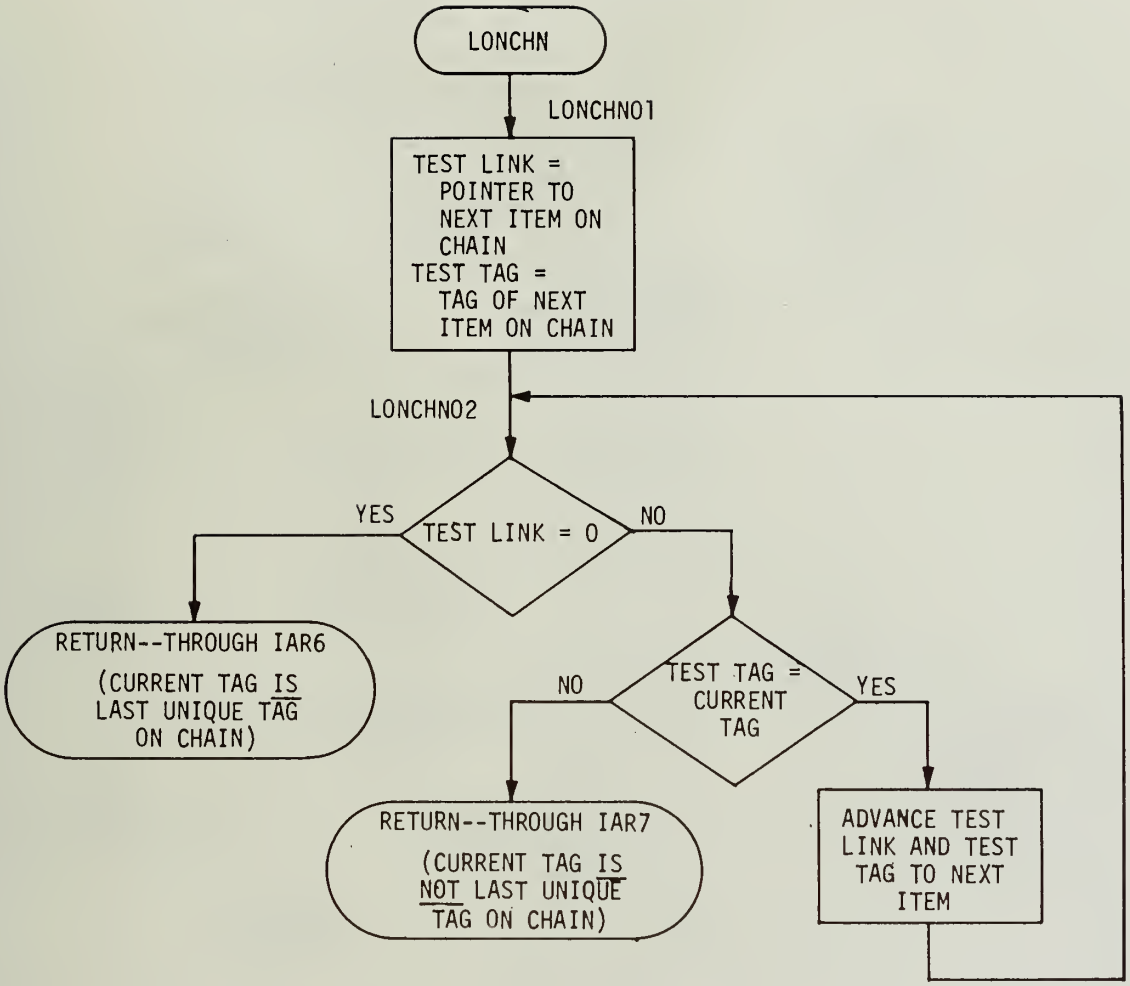


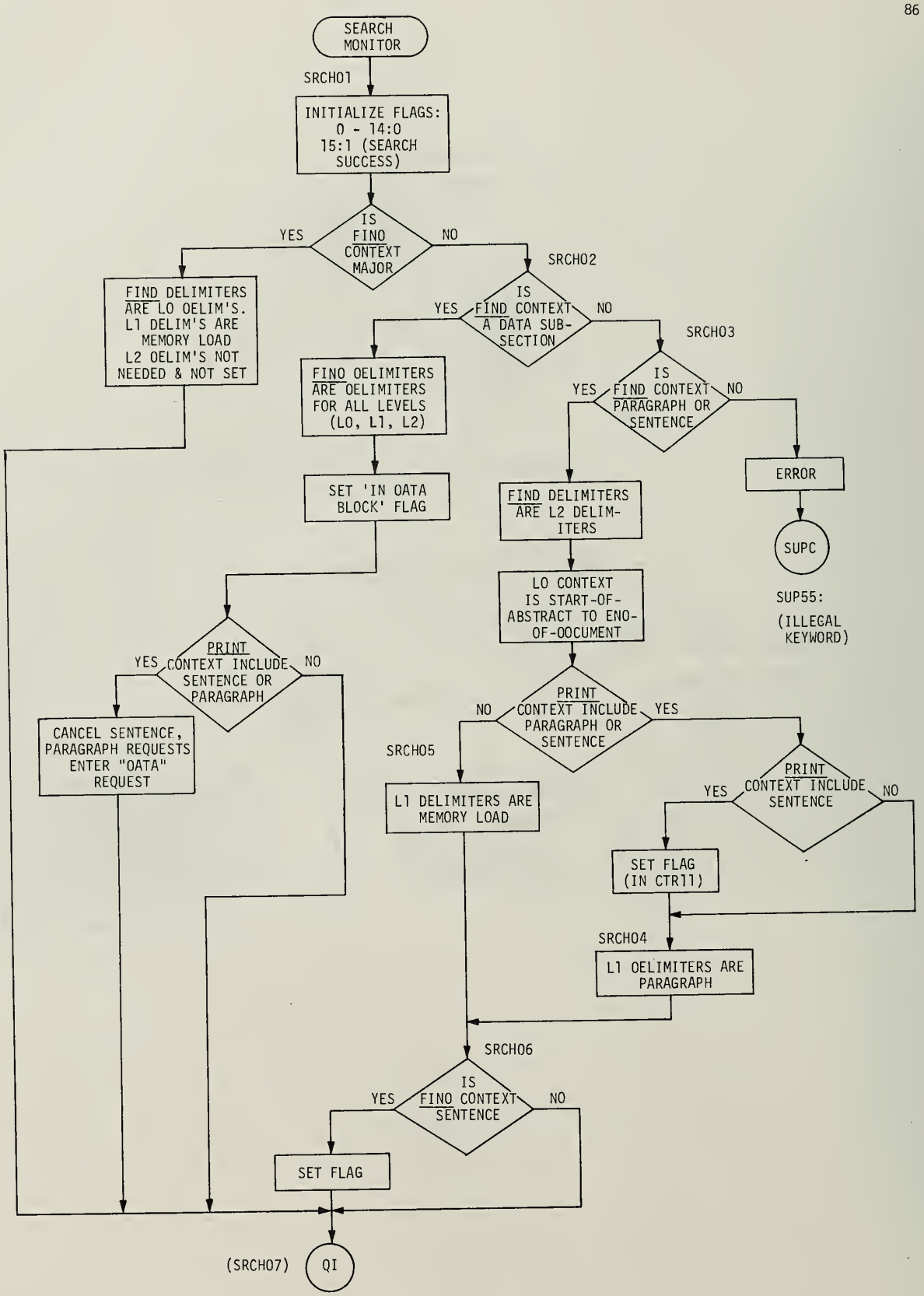
NOTES: INPUTS: LOW PTR = ADDRESS OF FIRST ELEMENT IN SORT LIST
HIGH PTR = ADDRESS OF LAST ELEMENT IN SORT LIST
STEP = ADDRESS INCREMENT BETWEEN ELEMENTS

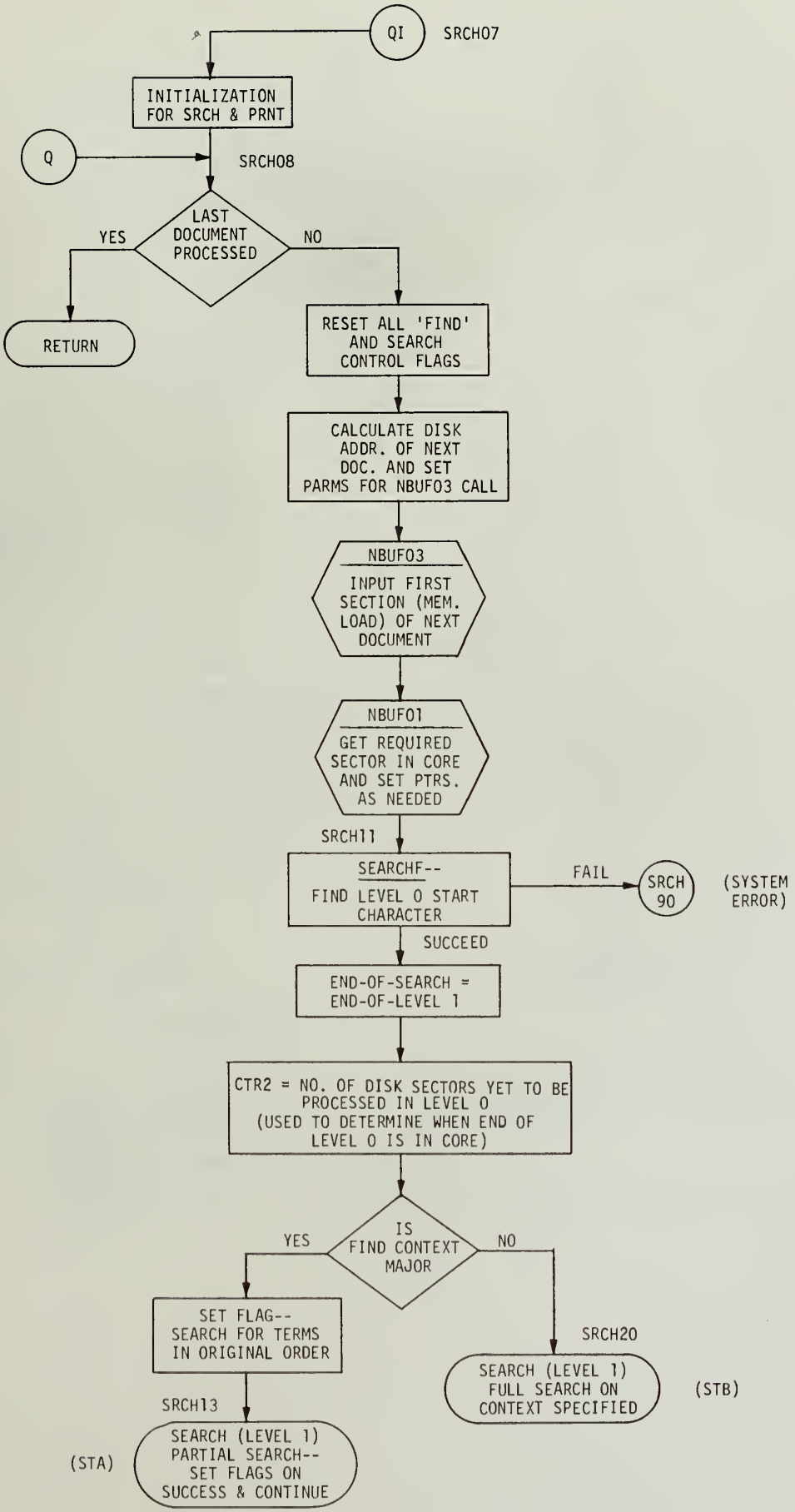
EACH "ELEMENT" IS A POINTER TO SORT KEY. ELEMENTS, BUT NOT SORT RECORDS, GET PHYSICALLY REARRANGED IN MEMORY.

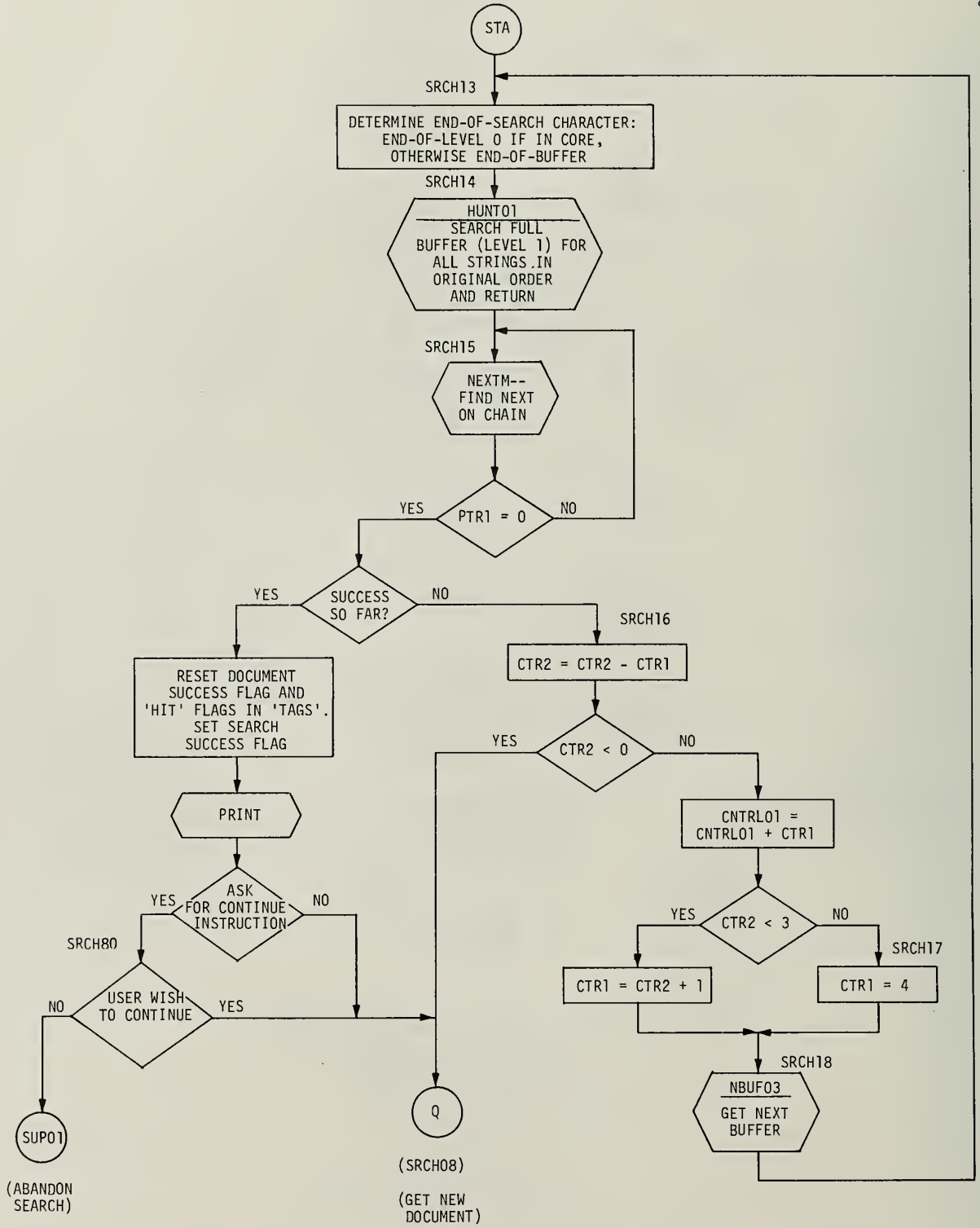
OUTPUT: LIST OF ELEMENTS ARRANGED IN ORDER OF INCREASING SORT KEY VALUES

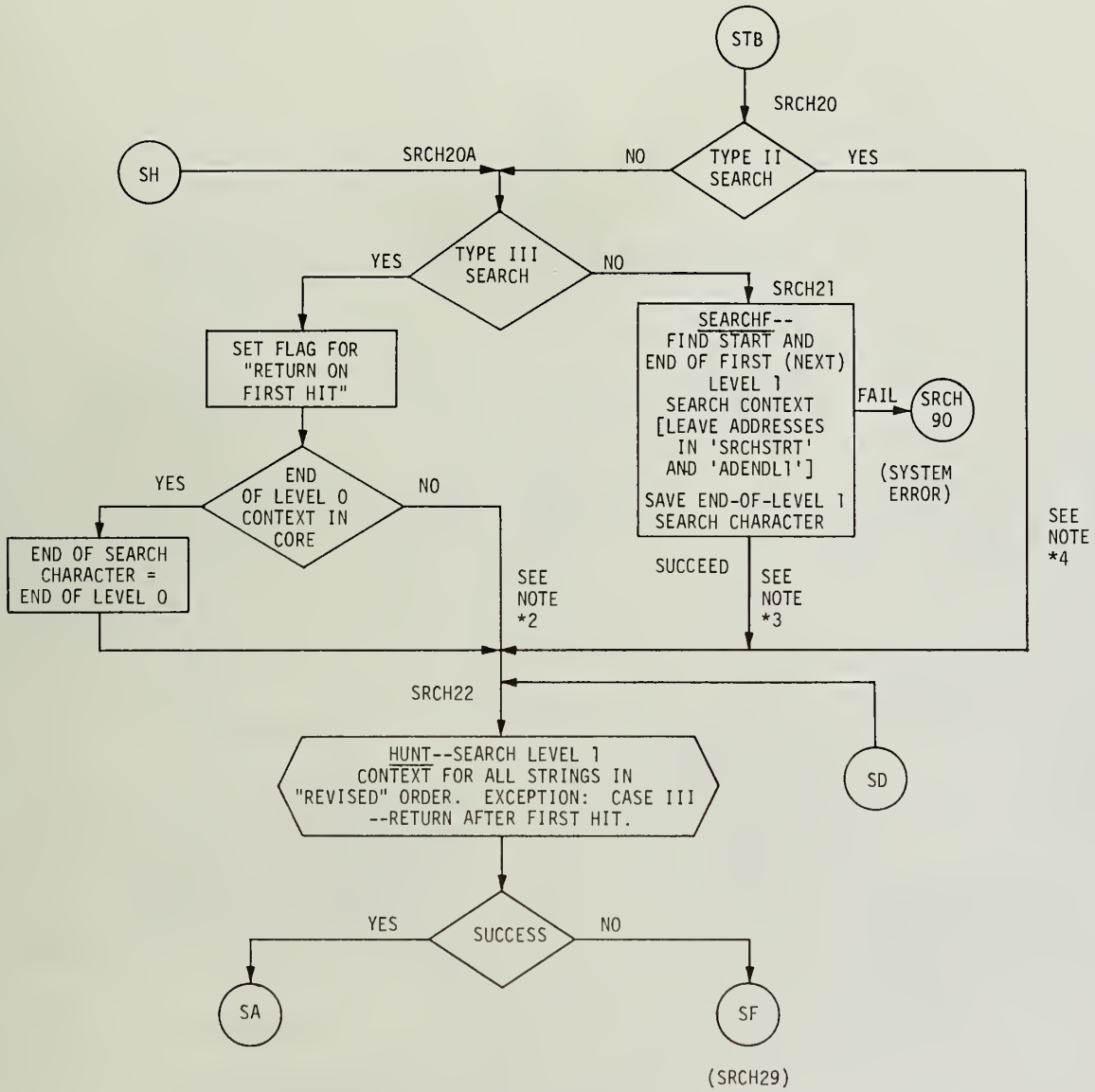

LWTAG
LWTAG01
IS CURRENT ENTITY AT END OF CHAIN
YES
NO
CURRENT TAG = NEXT TAG ON CHAIN
NO
YES
RETURN THROUGH IAR6 (CURRENT ENTITY IS LAST WITH CURRENT TAG)
RETURN THROUGH IAR7 (CURRENT ENTITY IS NOT LAST WITH CURRENT TAG)


LONCHN
LONCHN01
TEST LINK = POINTER TO NEXT ITEM ON CHAIN
TEST TAG = TAG OF NEXT ITEM ON CHAIN
LONCHN02
TEST LINK = 0
YES
NO
RETURN--THROUGH IAR6
(CURRENT TAG IS LAST UNIQUE TAG ON CHAIN)
TEST TAG = CURRENT TAG
NO
YES
RETURN--THROUGH IAR7
(CURRENT TAG IS NOT LAST UNIQUE TAG ON CHAIN)
ADVANCE TEST LINK AND TEST TAG TO NEXT ITEM


SEARCH MONITOR
SRCH01
INITIALIZE FLAGS: 0 - 14:0 15:1 (SEARCH SUCCESS)
IS FIND CONTEXT MAJOR
YES
NO
FIND DELIMITERS ARE L0 DELIM'S. L1 DELIM'S ARE MEMORY LOAD L2 DELIM'S NOT NEEDED & NOT SET
SRCH02
IS FIND CONTEXT A DATA SUB-SECTION
YES
NO
FIND DELIMITERS ARE DELIMITERS FOR ALL LEVELS (L0, L1, L2)
SET 'IN DATA BLOCK' FLAG
SRCH03
IS FIND CONTEXT PARAGRAPH OR SENTENCE
YES
NO
ERROR
SUPC
SUP55: (ILLEGAL KEYWORD)
FIND DELIMITERS ARE L2 DELIM-ITERS
L0 CONTEXT IS START-OF-ABSTRACT TO END-OF-DOCUMENT
PRINT CONTEXT INCLUDE SENTENCE OR PARAGRAPH
YES
NO
CANCEL SENTENCE, PARAGRAPH REQUESTS ENTER "DATA" REQUEST
PRINT CONTEXT INCLUDE PARAGRAPH OR SENTENCE
NO
YES
SRCH05
L1 DELIMITERS ARE MEMORY LOAD
PRINT CONTEXT INCLUDE SENTENCE
YES
NO
SET FLAG (IN CTR11)
SRCH04
L1 DELIMITERS ARE PARAGRAPH
SRCH06
IS FIND CONTEXT SENTENCE
YES
NO
SET FLAG
(SRCH07)
QI


QI
SRCH07
INITIALIZATION FOR SRCH & PRNT
Q
SRCH08
LAST DOCUMENT PROCESSED
YES
NO
RETURN
RESET ALL 'FIND' AND SEARCH CONTROL FLAGS
CALCULATE DISK ADDR. OF NEXT DOC. AND SET PARMS FOR NBUF03 CALL
NBUF03
INPUT FIRST SECTION (MEM. LOAD) OF NEXT DOCUMENT
NBUF01
GET REQUIRED SECTOR IN CORE AND SET PTRS. AS NEEDED
SRCH11
SEARCHF--
FIND LEVEL 0 START CHARACTER
FAIL
SRCH 90
(SYSTEM ERROR)
SUCCEED
END-OF-SEARCH = END-OF-LEVEL 1
CTR2 = NO. OF DISK SECTORS YET TO BE PROCESSED IN LEVEL 0 (USED TO DETERMINE WHEN END OF LEVEL 0 IS IN CORE)
IS FIND CONTEXT MAJOR
YES
NO
SET FLAG-- SEARCH FOR TERMS IN ORIGINAL ORDER
SRCH13
SEARCH (LEVEL 1) PARTIAL SEARCH-- SET FLAGS ON SUCCESS & CONTINUE
(STA)
SRCH20
SEARCH (LEVEL 1) FULL SEARCH ON CONTEXT SPECIFIED
(STB)


STA
SRCH13
DETERMINE END-OF-SEARCH CHARACTER: END-OF-LEVEL 0 IF IN CORE, OTHERWISE END-OF-BUFFER
SRCH14
HUNT01
SEARCH FULL BUFFER (LEVEL 1) FOR ALL STRINGS IN ORIGINAL ORDER AND RETURN
SRCH15
NEXTM-- FIND NEXT ON CHAIN
PTR1 = 0
YES
NO
SUCCESS SO FAR?
YES
NO
SRCH16
CTR2 = CTR2 - CTR1
RESET DOCUMENT SUCCESS FLAG AND 'HIT' FLAGS IN 'TAGS'. SET SEARCH SUCCESS FLAG
CTR2 < 0
YES
NO
PRINT
CNTRL01 = CNTRL01 + CTR1
ASK FOR CONTINUE INSTRUCTION
YES
NO
CTR2 < 3
YES
NO
SRCH17
SRCH80
USER WISH TO CONTINUE
NO
YES
CTR1 = CTR2 + 1
CTR1 = 4
SRCH18
NBUF03
GET NEXT BUFFER
SUP01
(ABANDON SEARCH)
Q
(SRCH08)
(GET NEW DOCUMENT)



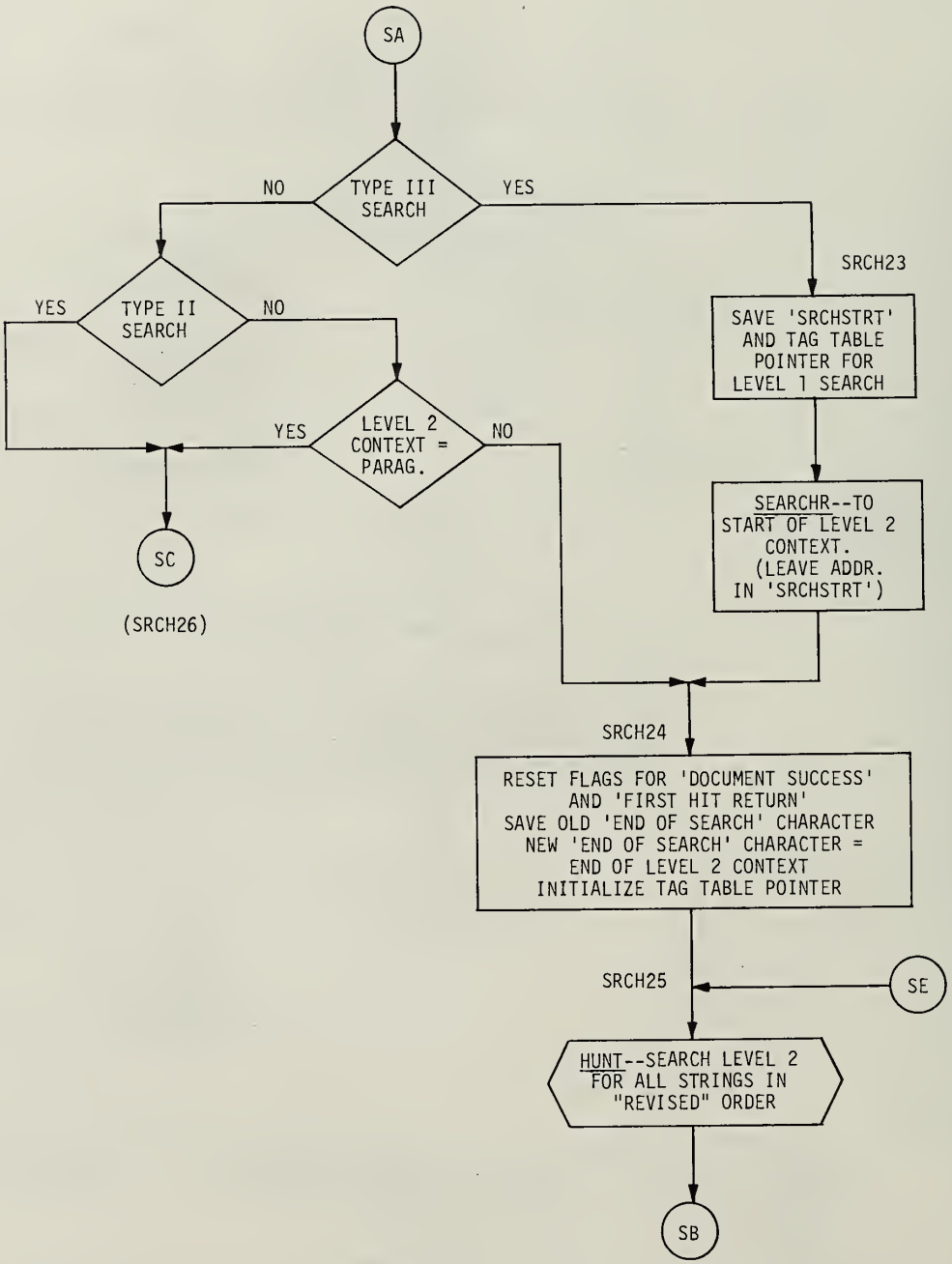
NOTES:

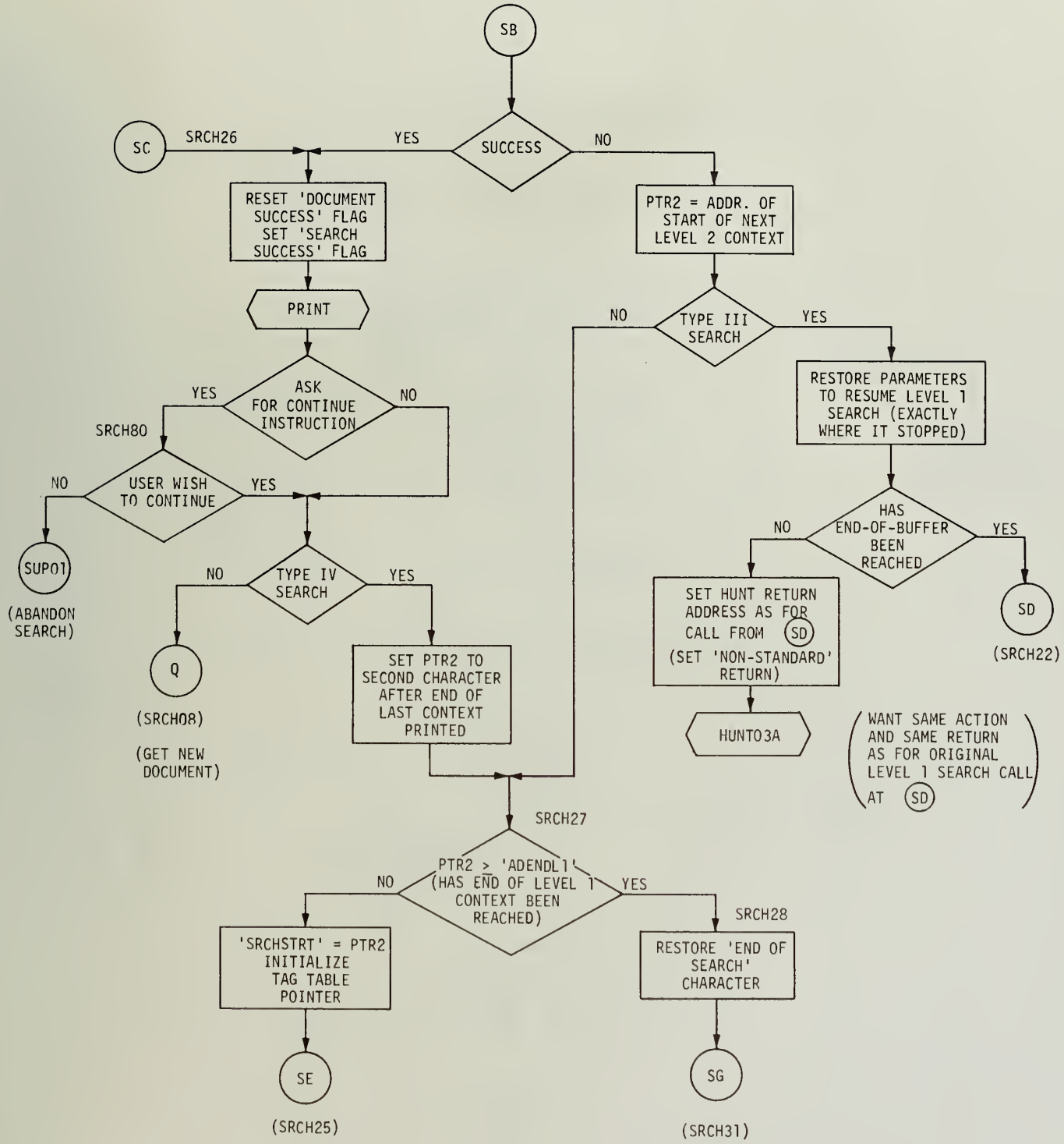
1. CASE III, LEVEL 1 AND CASE IV, LEVELS 1 AND 2 REQUIRE BUFFER TO END ON PARAGRAPH OR SENTENCE DELIMITERS. ALWAYS END TEXT BUFFER WITH :88EEEC:, I.E., DEFINE END OF BUFFER TO BE BOTH END OF PARAGRAPH AND END OF SENTENCE.

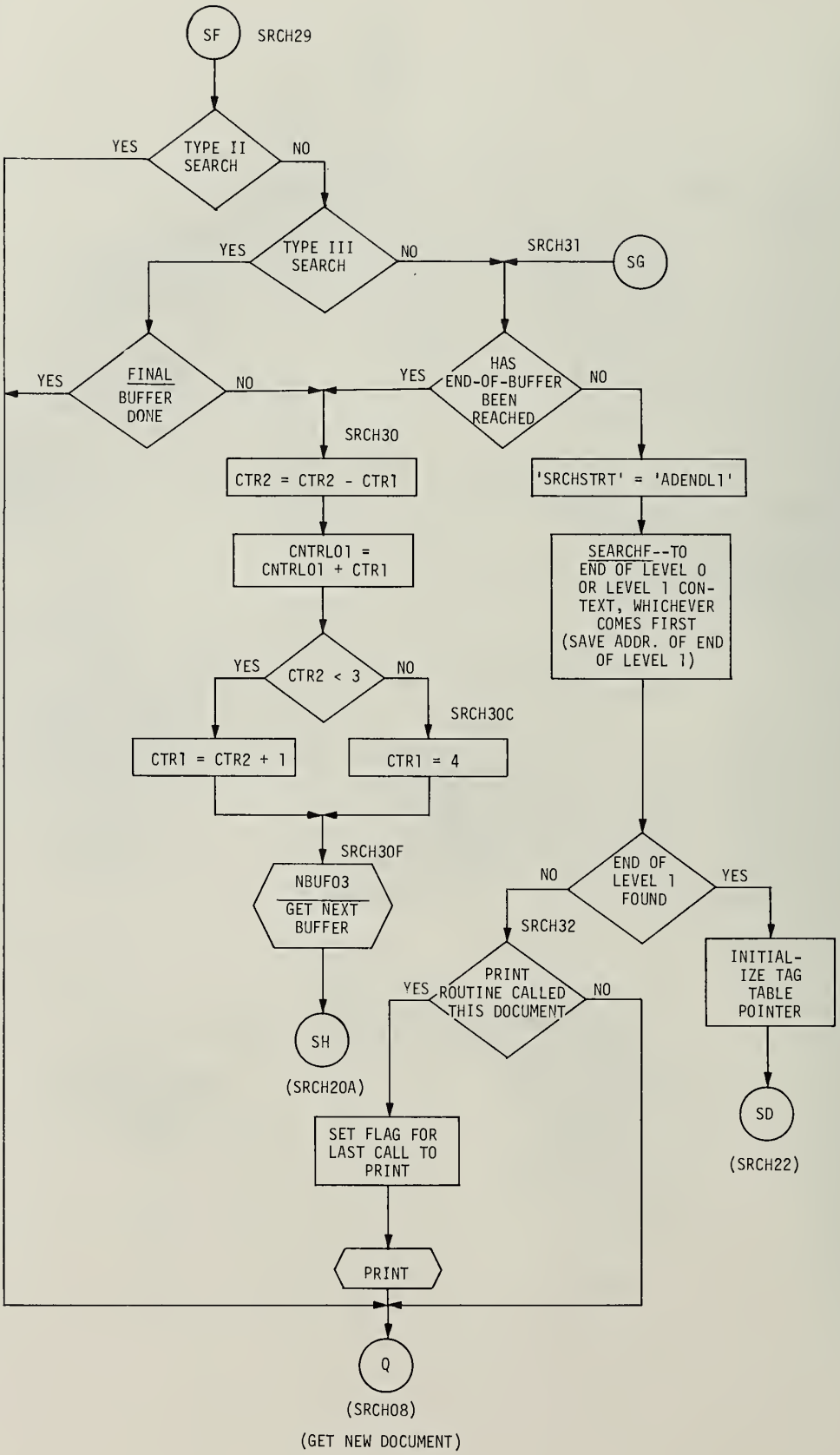
*2. END OF SEARCH = END OF BUFFER (ALREADY SET).

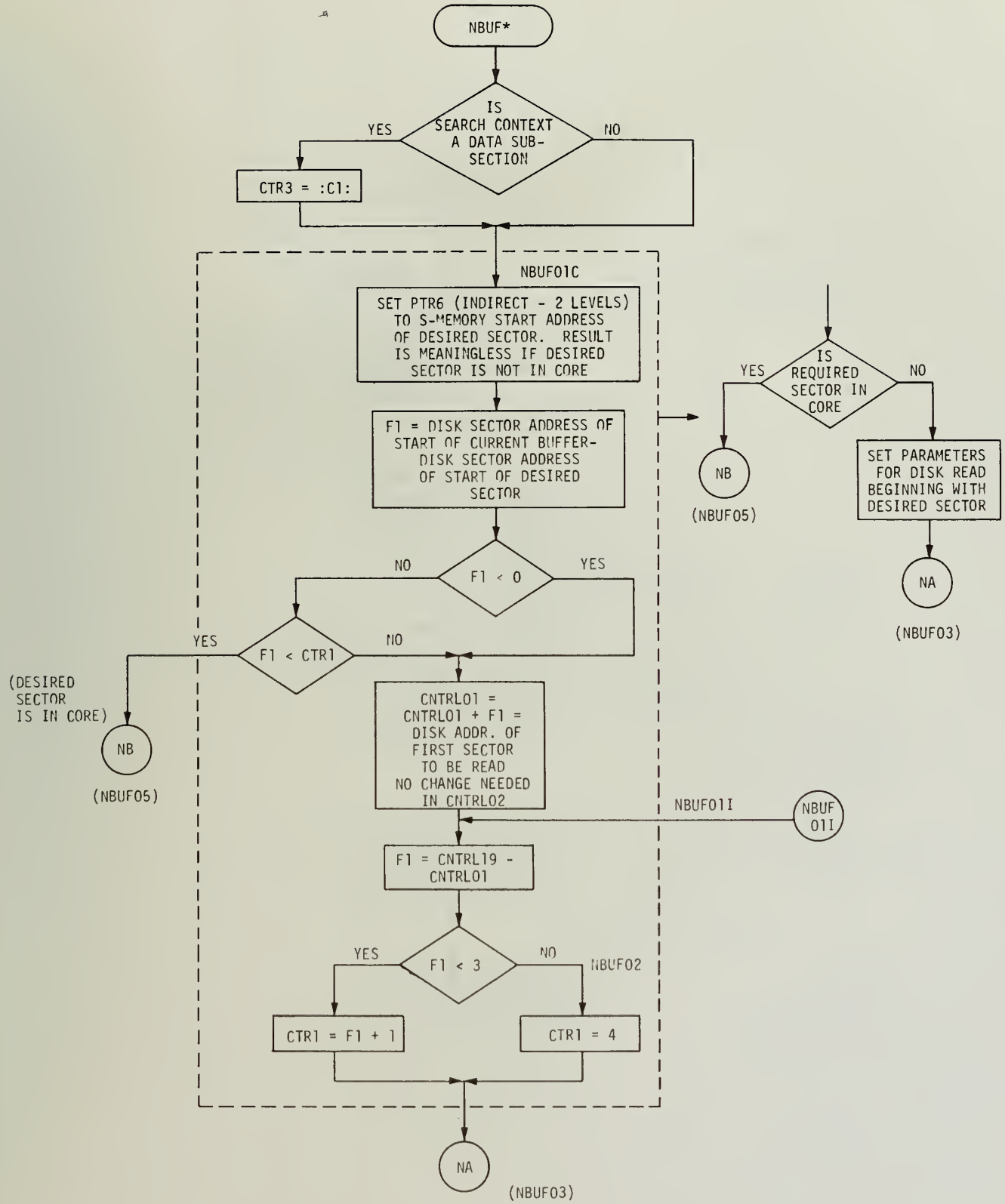
*3. END OF SEARCH = END OF LEVEL 1 CONTEXT = PARAGRAPH DELIMITER (ALREADY SET).

*4. END OF SEARCH = END OF LEVEL 1 CONTEXT = END OF FIND CONTEXT (ALREADY SET).


SA
TYPE III SEARCH
NO
YES
TYPE II SEARCH
YES
NO
LEVEL 2 CONTEXT = PARAG.
YES
NO
SC
(SRCH26)
SRCH23
SAVE 'SRCHSTRT' AND TAG TABLE POINTER FOR LEVEL 1 SEARCH
SEARCHR--TO START OF LEVEL 2 CONTEXT. (LEAVE ADDR. IN 'SRCHSTRT')
SRCH24
RESET FLAGS FOR 'DOCUMENT SUCCESS' AND 'FIRST HIT RETURN'
SAVE OLD 'END OF SEARCH' CHARACTER
NEW 'END OF SEARCH' CHARACTER = END OF LEVEL 2 CONTEXT
INITIALIZE TAG TABLE POINTER
SRCH25
SE
HUNT--SEARCH LEVEL 2 FOR ALL STRINGS IN "REVISED" ORDER
SB


SB
SUCCESS
YES
NO
SC
SRCH26
RESET 'DOCUMENT SUCCESS' FLAG
SET 'SEARCH SUCCESS' FLAG
PRINT
ASK FOR CONTINUE INSTRUCTION
YES
NO
SRCH80
USER WISH TO CONTINUE
NO
YES
SUP01
(ABANDON SEARCH)
TYPE IV SEARCH
NO
YES
Q
(SRCH08)
(GET NEW DOCUMENT)
SET PTR2 TO SECOND CHARACTER AFTER END OF LAST CONTEXT PRINTED
PTR2 = ADDR. OF START OF NEXT LEVEL 2 CONTEXT
NO
TYPE III SEARCH
YES
RESTORE PARAMETERS TO RESUME LEVEL 1 SEARCH (EXACTLY WHERE IT STOPPED)
HAS END-OF-BUFFER BEEN REACHED
NO
YES
SD
(SRCH22)
SET HUNT RETURN ADDRESS AS FOR CALL FROM (SD)
(SET 'NON-STANDARD' RETURN)
HUNT03A
(WANT SAME ACTION AND SAME RETURN AS FOR ORIGINAL LEVEL 1 SEARCH CALL AT (SD))
SRCH27
PTR2 ≥ 'ADENDL1' (HAS END OF LEVEL 1 CONTEXT BEEN REACHED)
NO
YES
'SRCHSTRT' = PTR2
INITIALIZE TAG TABLE POINTER
SE
(SRCH25)
SRCH28
RESTORE 'END OF SEARCH' CHARACTER
SG
(SRCH31)


SF
SRCH29
TYPE II SEARCH
YES
NO
TYPE III SEARCH
YES
NO
SRCH31
SG
FINAL BUFFER DONE
YES
NO
HAS END-OF-BUFFER BEEN REACHED
YES
NO
SRCH30
CTR2 = CTR2 - CTR1
CNTRL01 = CNTRL01 + CTR1
CTR2 < 3
YES
NO
SRCH30C
CTR1 = CTR2 + 1
CTR1 = 4
SRCH30F
NBUF03
GET NEXT BUFFER
SH
(SRCH20A)
'SRCHSTRT' = 'ADENDL1'
SEARCHF--TO END OF LEVEL 0 OR LEVEL 1 CONTEXT, WHICHEVER COMES FIRST (SAVE ADDR. OF END OF LEVEL 1)
END OF LEVEL 1 FOUND
NO
YES
SRCH32
PRINT ROUTINE CALLED THIS DOCUMENT
YES
NO
INITIALIZE TAG TABLE POINTER
SD
(SRCH22)
SET FLAG FOR LAST CALL TO PRINT
PRINT
Q
(SRCH08)
(GET NEW DOCUMENT)

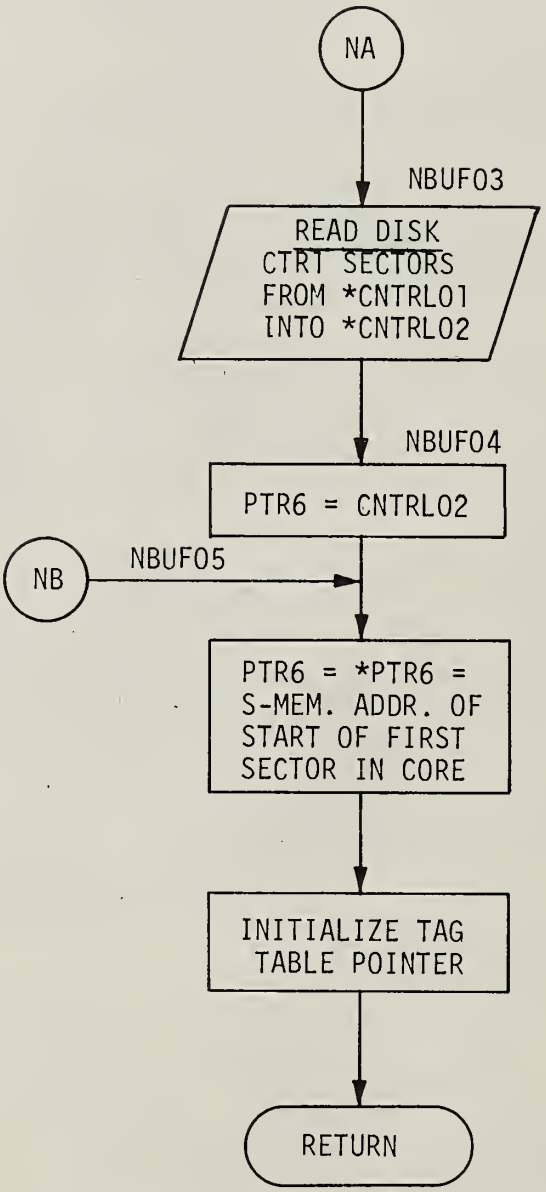


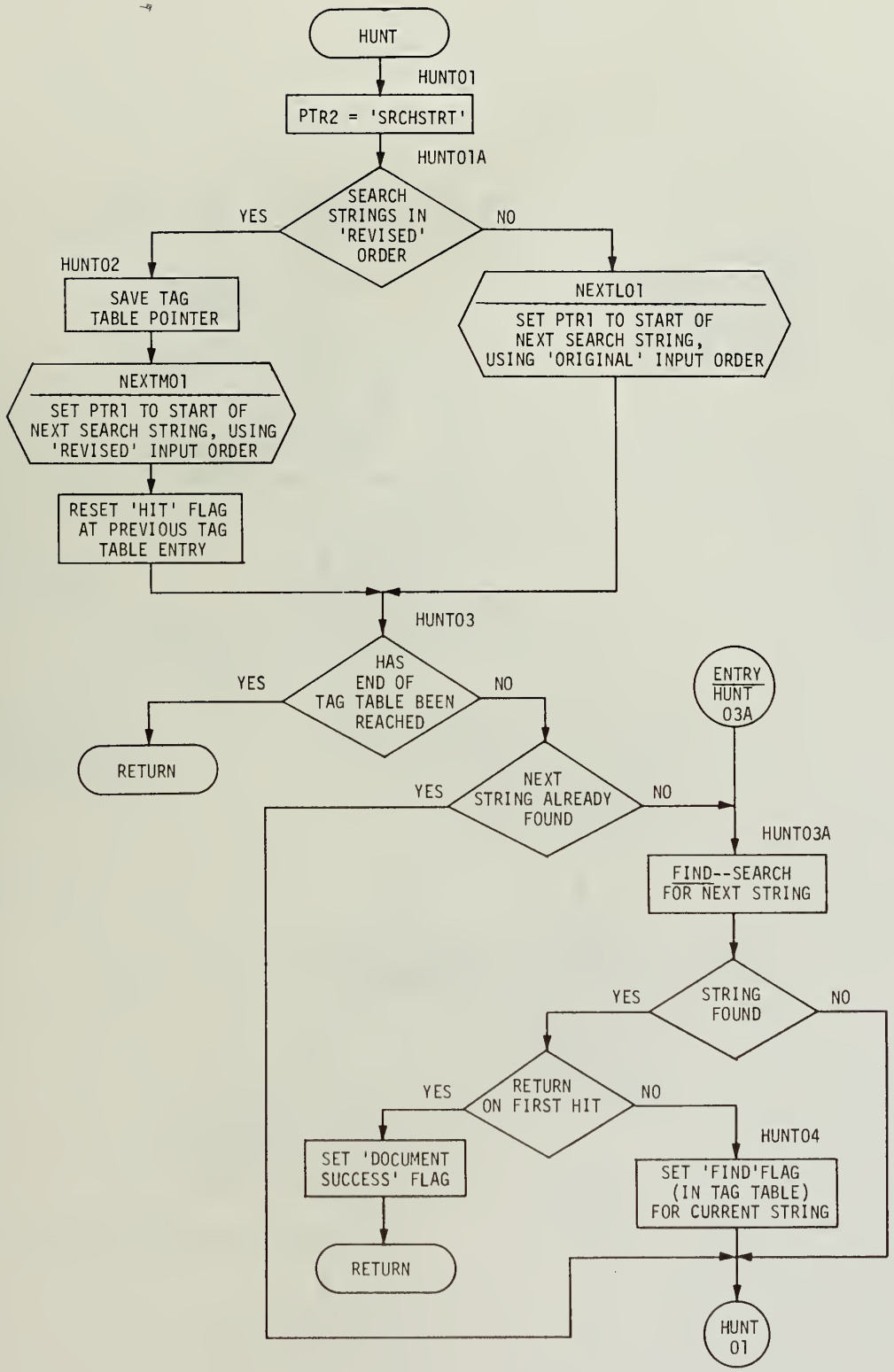
*CALLING CONVENTIONS:

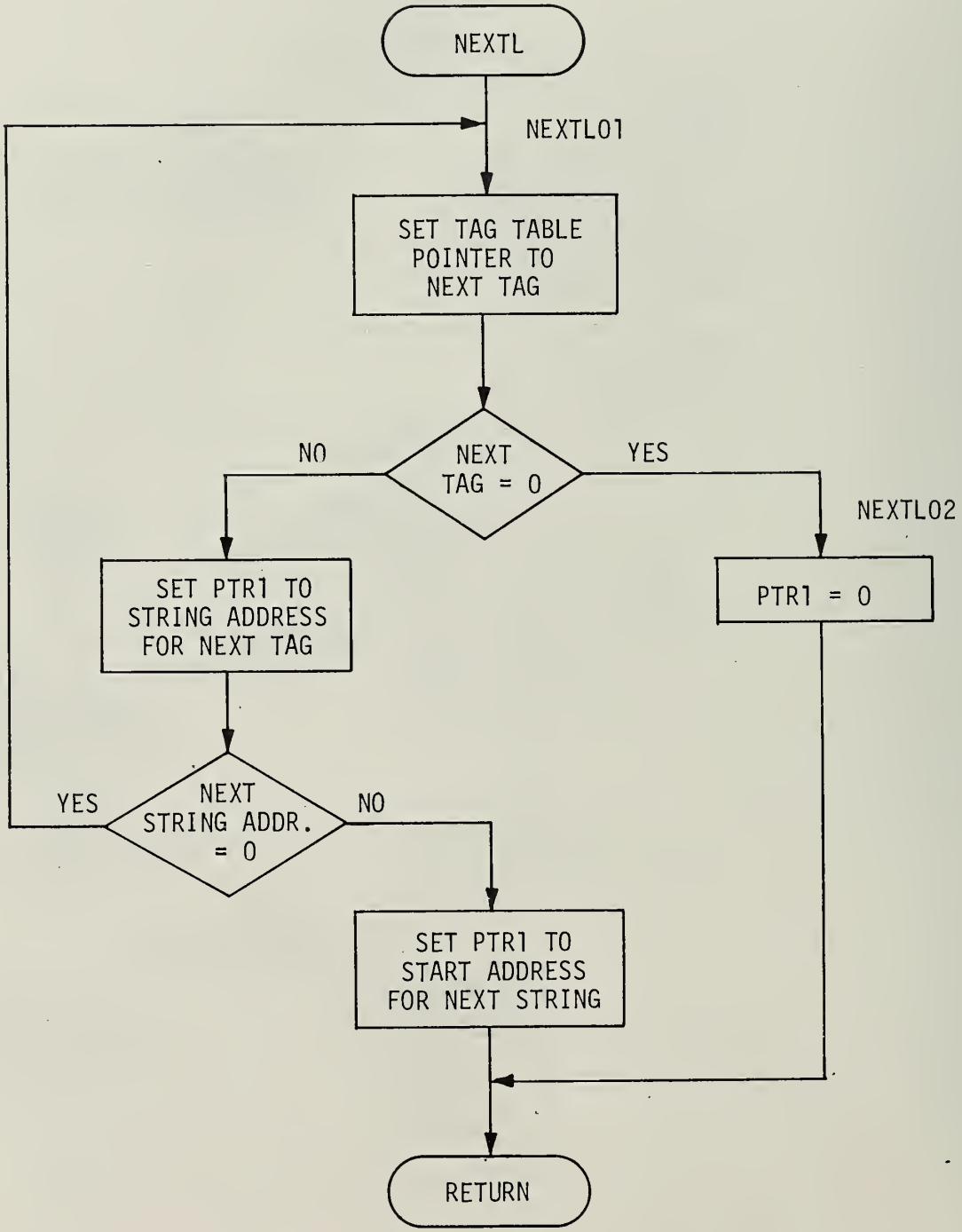
CTR1 = N = NUMBER OF SECTORS TO BE READ

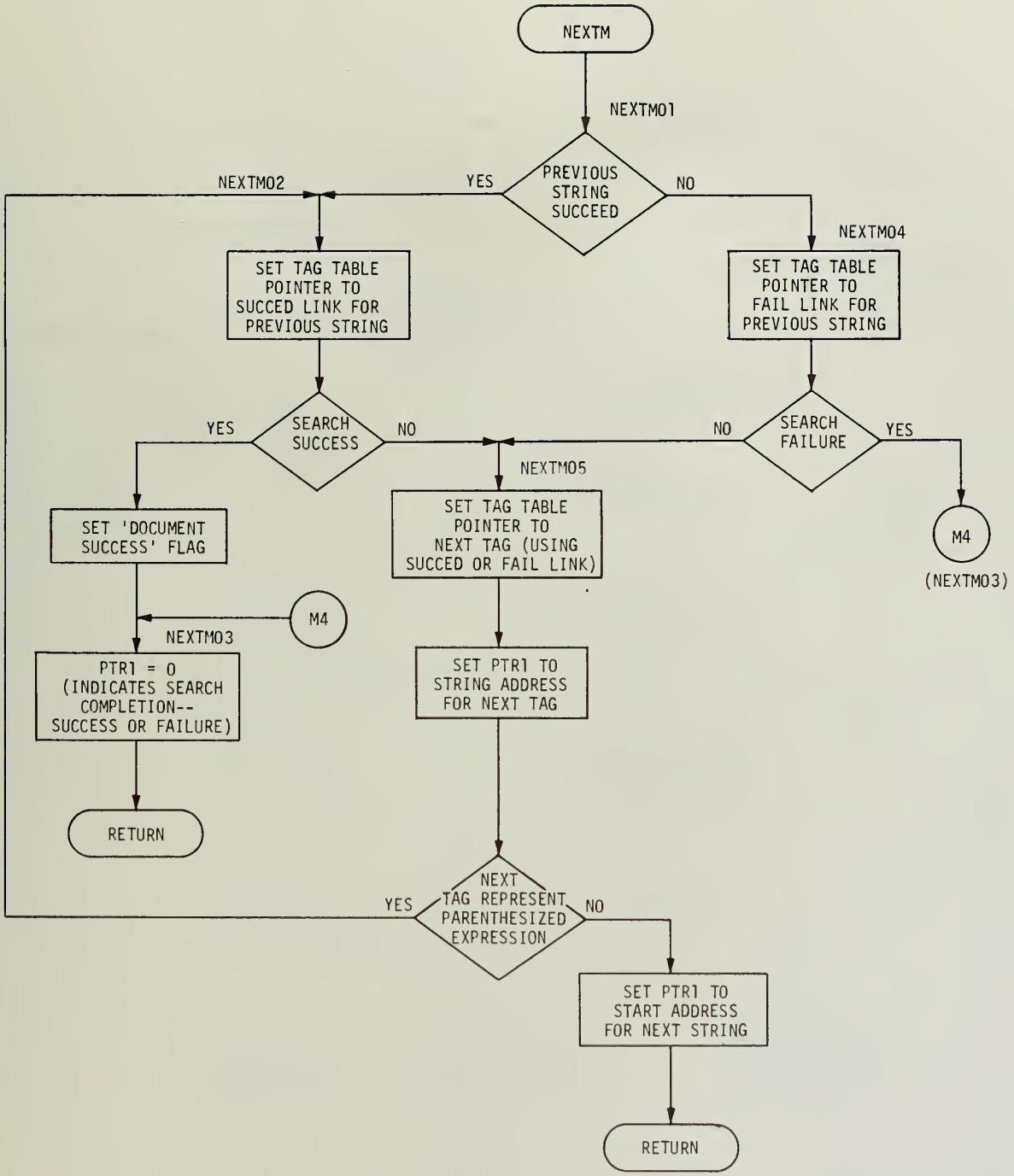
CNTRL01 = DISK ADDRESS OF FIPST SECTOR TO BE READ

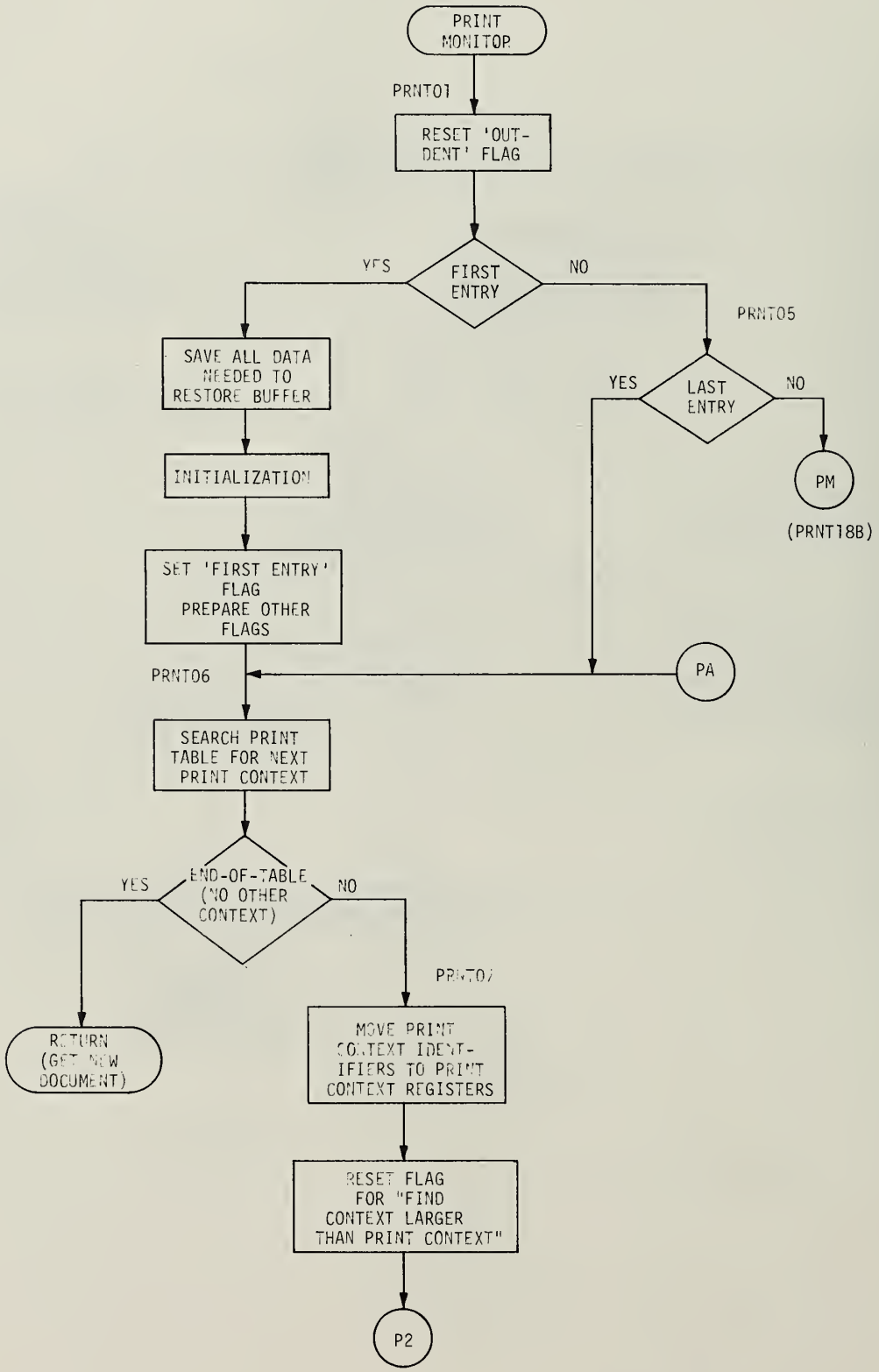
CNTRL02 = S-MEMORY ADDRESS FOR FIRST CHARACTER READ

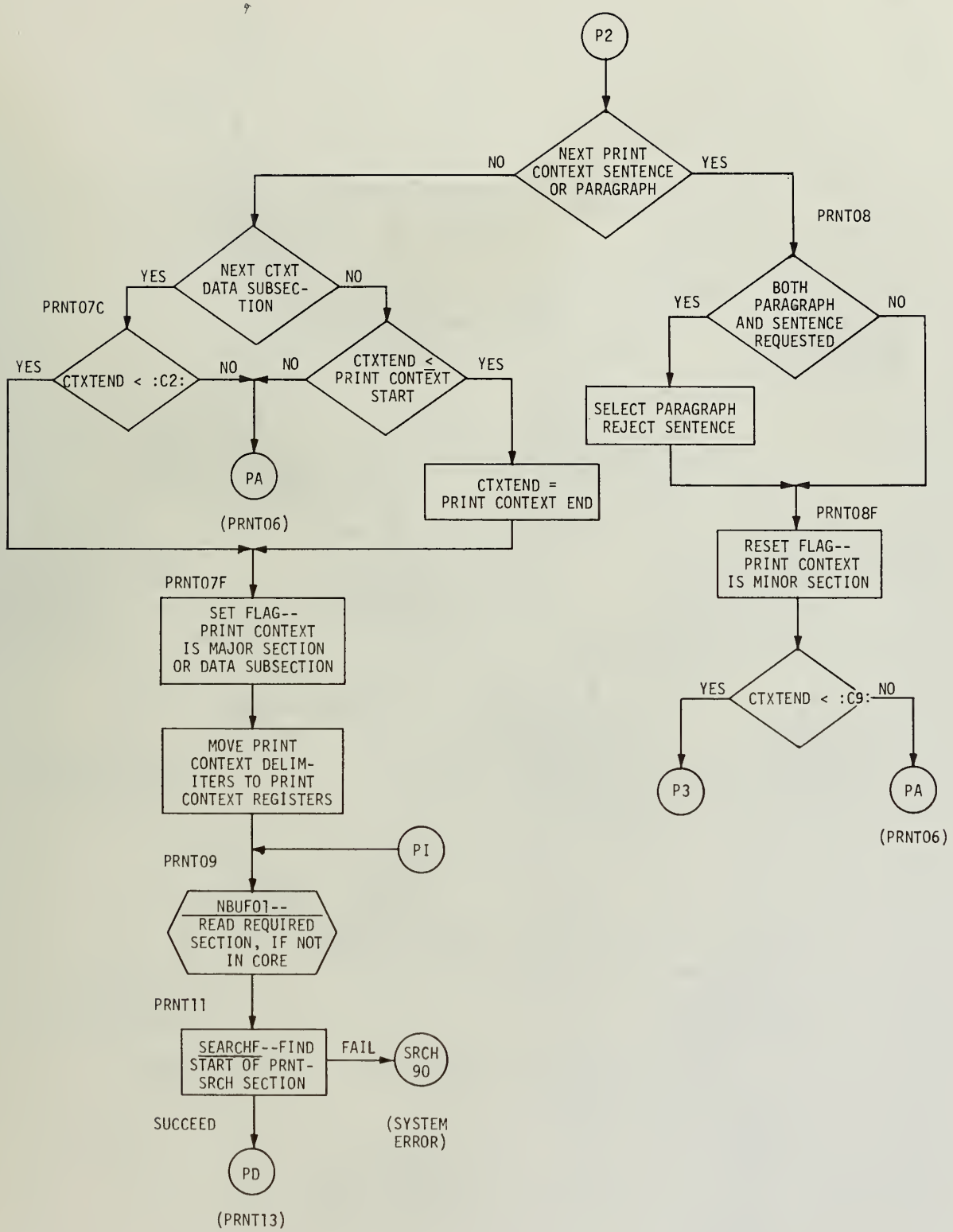

NA
NBUF03
READ DISK
CTRT SECTORS
FROM *CNTRL01
INTO *CNTRL02
NBUF04
PTR6 = CNTRL02
NB
NBUF05
PTR6 = *PTR6 =
S-MEM. ADDR. OF
START OF FIRST
SECTOR IN CORE
INITIALIZE TAG
TABLE POINTER
RETURN

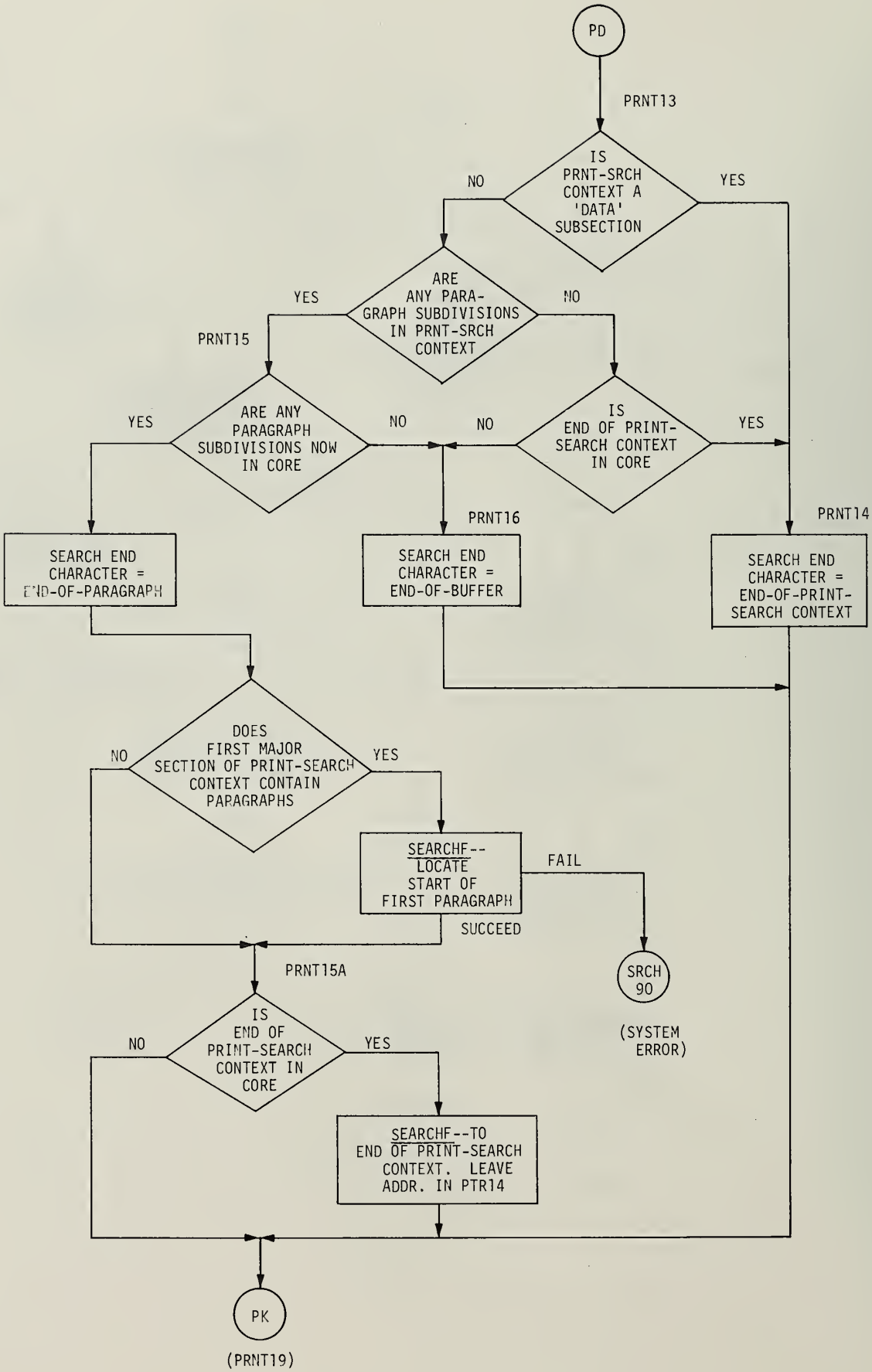

HUNT
HUNT01
PTR2 = 'SRCHSTRT'
HUNT01A
SEARCH STRINGS IN 'REVISED' ORDER
YES
NO
HUNT02
SAVE TAG TABLE POINTER
NEXTL01
SET PTR1 TO START OF NEXT SEARCH STRING, USING 'ORIGINAL' INPUT ORDER
NEXTM01
SET PTR1 TO START OF NEXT SEARCH STRING, USING 'REVISED' INPUT ORDER
RESET 'HIT' FLAG AT PREVIOUS TAG TABLE ENTRY
HUNT03
HAS END OF TAG TABLE BEEN REACHED
YES
NO
RETURN
ENTRY HUNT 03A
NEXT STRING ALREADY FOUND
YES
NO
HUNT03A
FIND--SEARCH FOR NEXT STRING
STRING FOUND
YES
NO
RETURN ON FIRST HIT
YES
NO
HUNT04
SET 'DOCUMENT SUCCESS' FLAG
SET 'FIND'FLAG (IN TAG TABLE) FOR CURRENT STRING
RETURN
HUNT 01

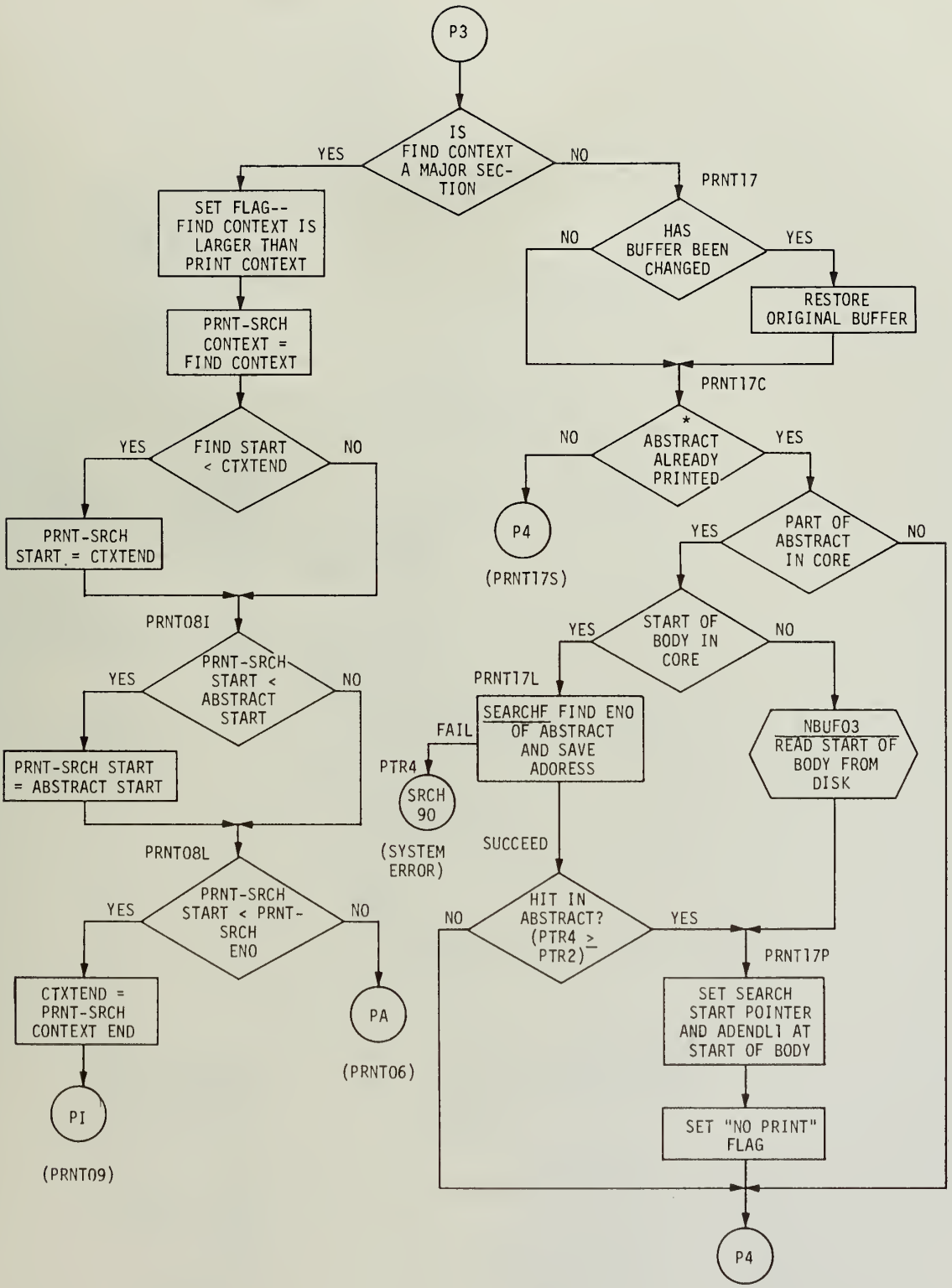

NEXTL
NEXTL01
SET TAG TABLE POINTER TO NEXT TAG
NEXT TAG = 0
NO
YES
NEXTL02
SET PTR1 TO STRING ADDRESS FOR NEXT TAG
PTR1 = 0
NEXT STRING ADDR. = 0
YES
NO
SET PTR1 TO START ADDRESS FOR NEXT STRING
RETURN


NEXTM
NEXTM01
PREVIOUS STRING SUCCEED
YES
NO
NEXTM02
SET TAG TABLE POINTER TO SUCCED LINK FOR PREVIOUS STRING
NEXTM04
SET TAG TABLE POINTER TO FAIL LINK FOR PREVIOUS STRING
SEARCH SUCCESS
YES
NO
SEARCH FAILURE
NO
YES
NEXTM05
SET 'DOCUMENT SUCCESS' FLAG
SET TAG TABLE POINTER TO NEXT TAG (USING SUCCED OR FAIL LINK)
M4
(NEXTM03)
M4
NEXTM03
PTR1 = 0 (INDICATES SEARCH COMPLETION-- SUCCESS OR FAILURE)
SET PTR1 TO STRING ADDRESS FOR NEXT TAG
RETURN
NEXT TAG REPRESENT PARENTHESIZED EXPRESSION
YES
NO
SET PTR1 TO START ADDRESS FOR NEXT STRING
RETURN

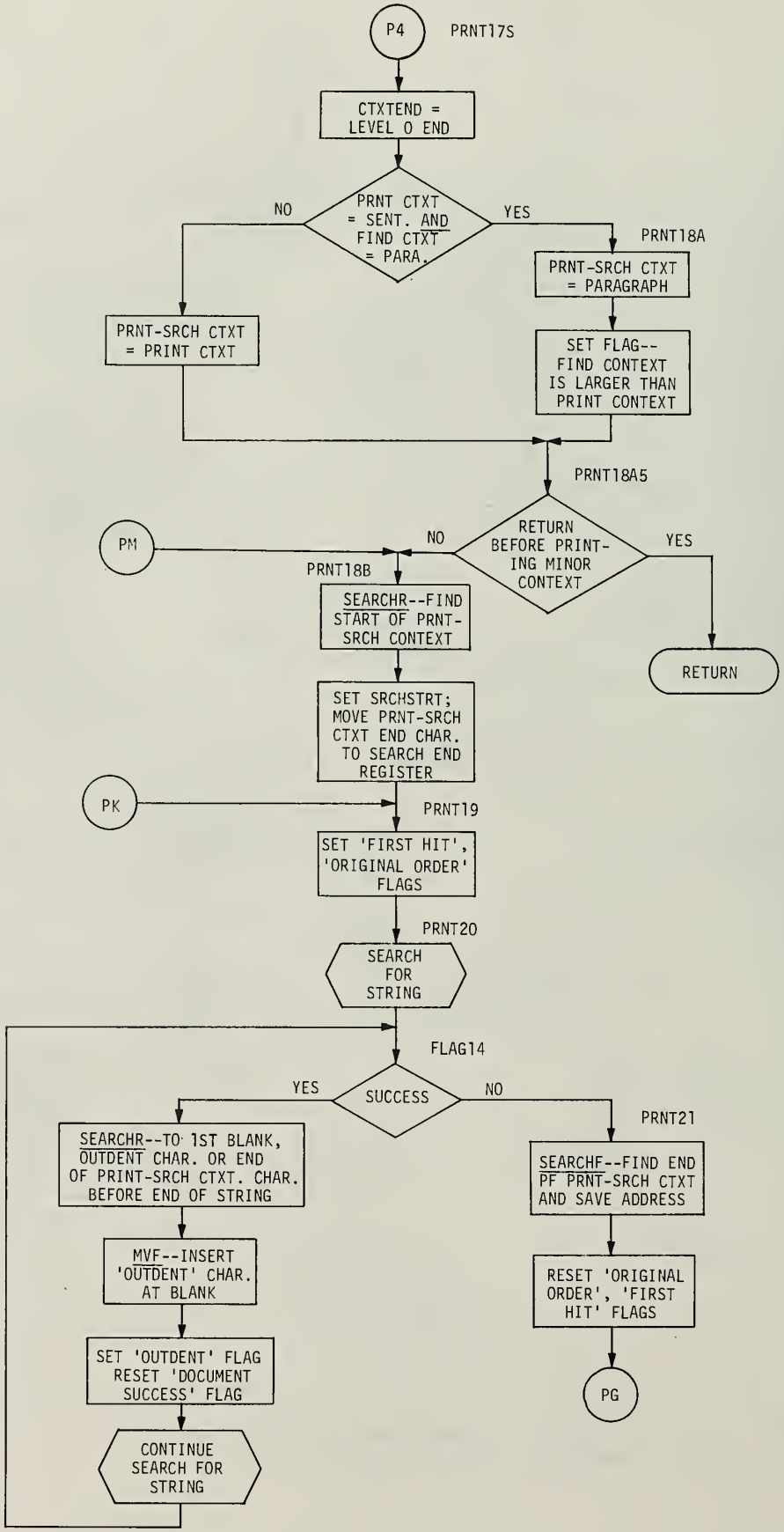

PRINT MONITOR
PRNT01
RESET 'OUT-DENT' FLAG
FIRST ENTRY
YES
NO
SAVE ALL DATA NEEDED TO RESTORE BUFFER
INITIALIZATION
SET 'FIRST ENTRY' FLAG
PREPARE OTHER FLAGS
PRNT05
LAST ENTRY
YES
NO
PM
(PRNT18B)
PA
PRNT06
SEARCH PRINT TABLE FOR NEXT PRINT CONTEXT
END-OF-TABLE (NO OTHER CONTEXT)
YES
NO
RETURN (GET NEW DOCUMENT)
PRNT07
MOVE PRINT CONTEXT IDENT-IFIERS TO PRINT CONTEXT REGISTERS
RESET FLAG FOR "FIND CONTEXT LARGER THAN PRINT CONTEXT"
P2

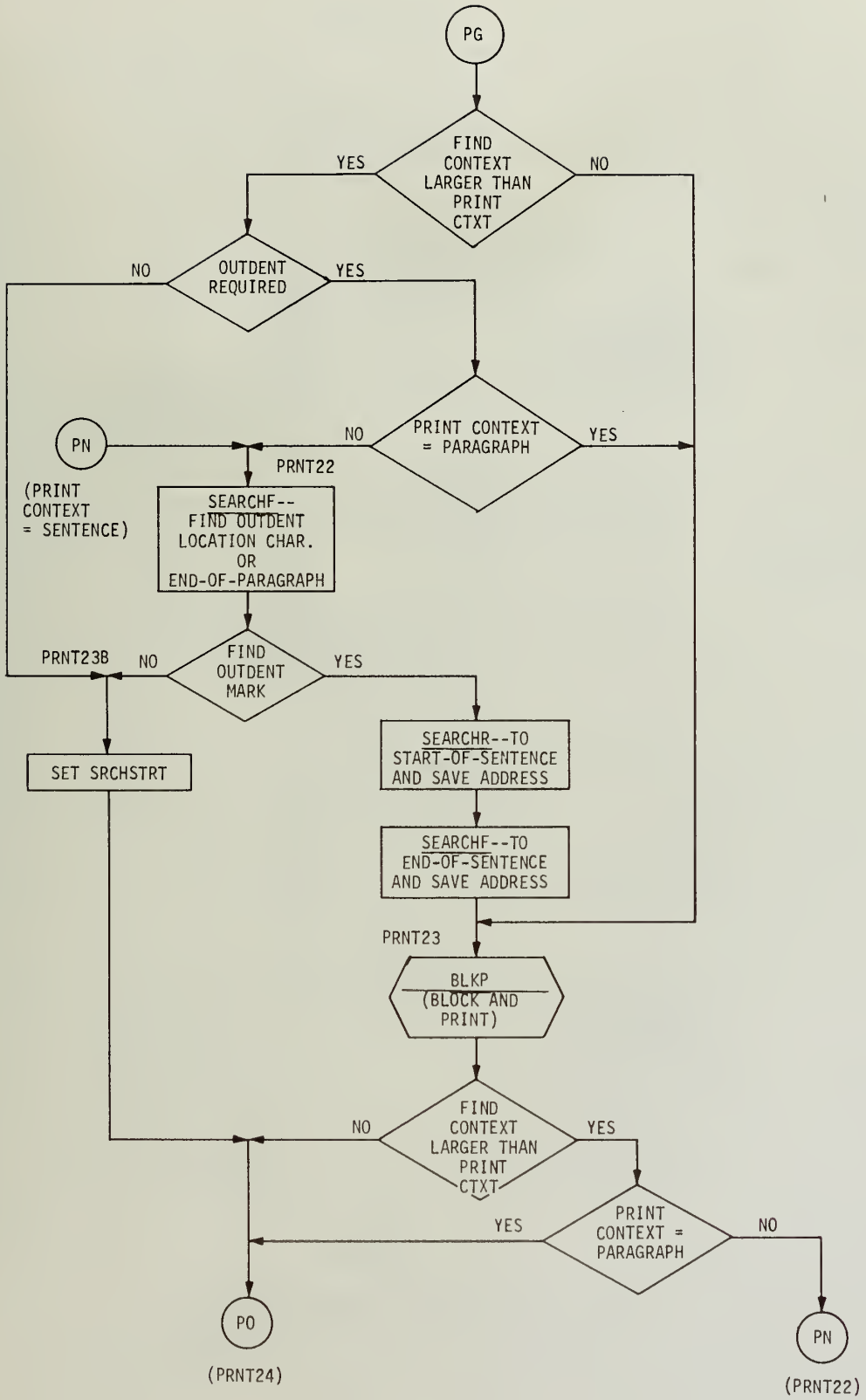

P2
NEXT PRINT CONTEXT SENTENCE OR PARAGRAPH
NO
YES
PRNT08
BOTH PARAGRAPH AND SENTENCE REQUESTED
YES
NO
SELECT PARAGRAPH REJECT SENTENCE
PRNT08F
RESET FLAG-- PRINT CONTEXT IS MINOR SECTION
CTXTEND < :C9:
YES
NO
P3
PA
(PRNT06)
NEXT CTXT DATA SUBSEC-TION
YES
NO
PRNT07C
CTXTEND < :C2:
YES
NO
CTXTEND < PRINT CONTEXT START
NO
YES
PA
(PRNT06)
CTXTEND = PRINT CONTEXT END
PRNT07F
SET FLAG-- PRINT CONTEXT IS MAJOR SECTION OR DATA SUBSECTION
MOVE PRINT CONTEXT DELIM-ITERS TO PRINT CONTEXT REGISTERS
PI
PRNT09
NBUF01-- READ REQUIRED SECTION, IF NOT IN CORE
PRNT11
SEARCHF--FIND START OF PRNT-SRCH SECTION
FAIL
SRCH 90
(SYSTEM ERROR)
SUCCEED
PD
(PRNT13)

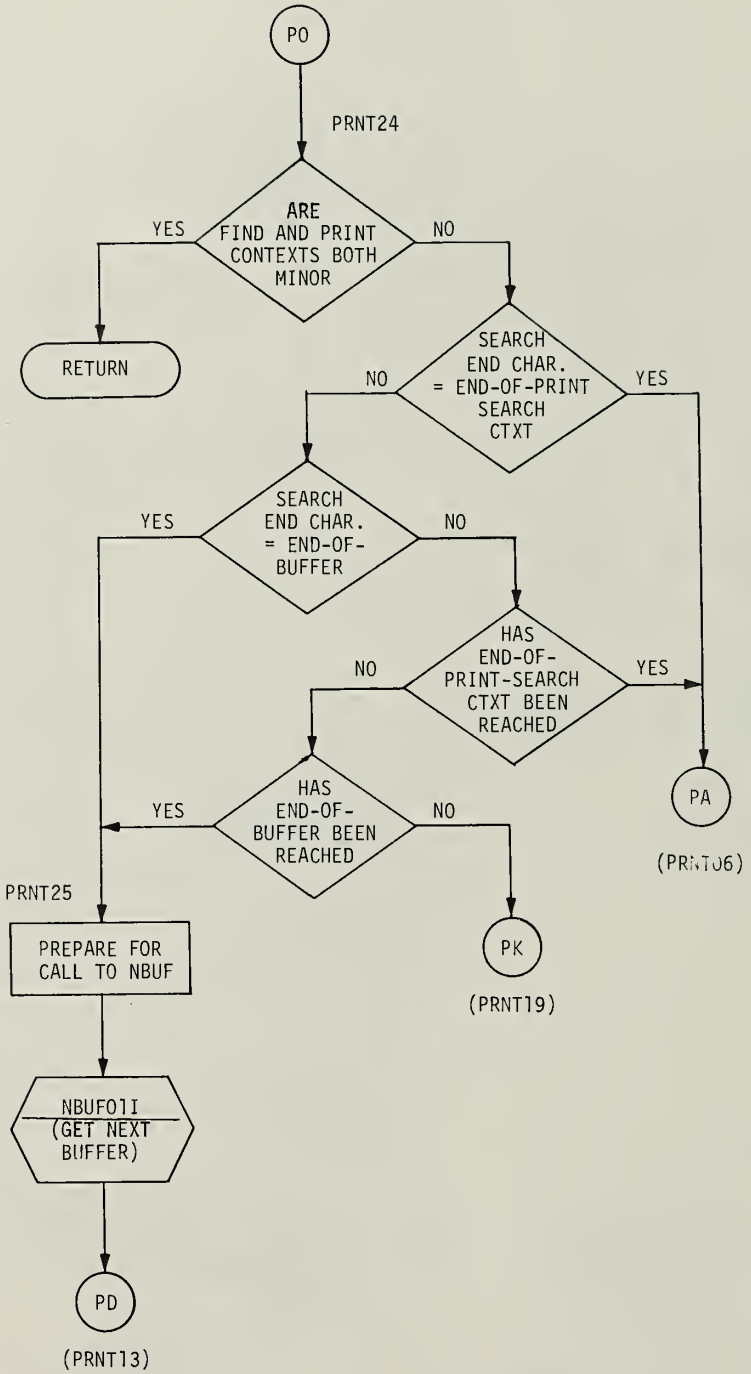

PD
PRNT13
IS PRNT-SRCH CONTEXT A 'DATA' SUBSECTION
NO
YES
ARE ANY PARA-GRAPH SUBDIVISIONS IN PRNT-SRCH CONTEXT
YES
NO
PRNT15
ARE ANY PARAGRAPH SUBDIVISIONS NOW IN CORE
YES
NO
NO
IS END OF PRINT-SEARCH CONTEXT IN CORE
YES
PRNT16
PRNT14
SEARCH END CHARACTER = END-OF-PARAGRAPH
SEARCH END CHARACTER = END-OF-BUFFER
SEARCH END CHARACTER = END-OF-PRINT-SEARCH CONTEXT
DOES FIRST MAJOR SECTION OF PRINT-SEARCH CONTEXT CONTAIN PARAGRAPHS
NO
YES
SEARCHF--LOCATE START OF FIRST PARAGRAPH
FAIL
SUCCEED
SRCH 90
(SYSTEM ERROR)
PRNT15A
IS END OF PRINT-SEARCH CONTEXT IN CORE
NO
YES
SEARCHF--TO END OF PRINT-SEARCH CONTEXT. LEAVE ADDR. IN PTR14
PK
(PRNT19)

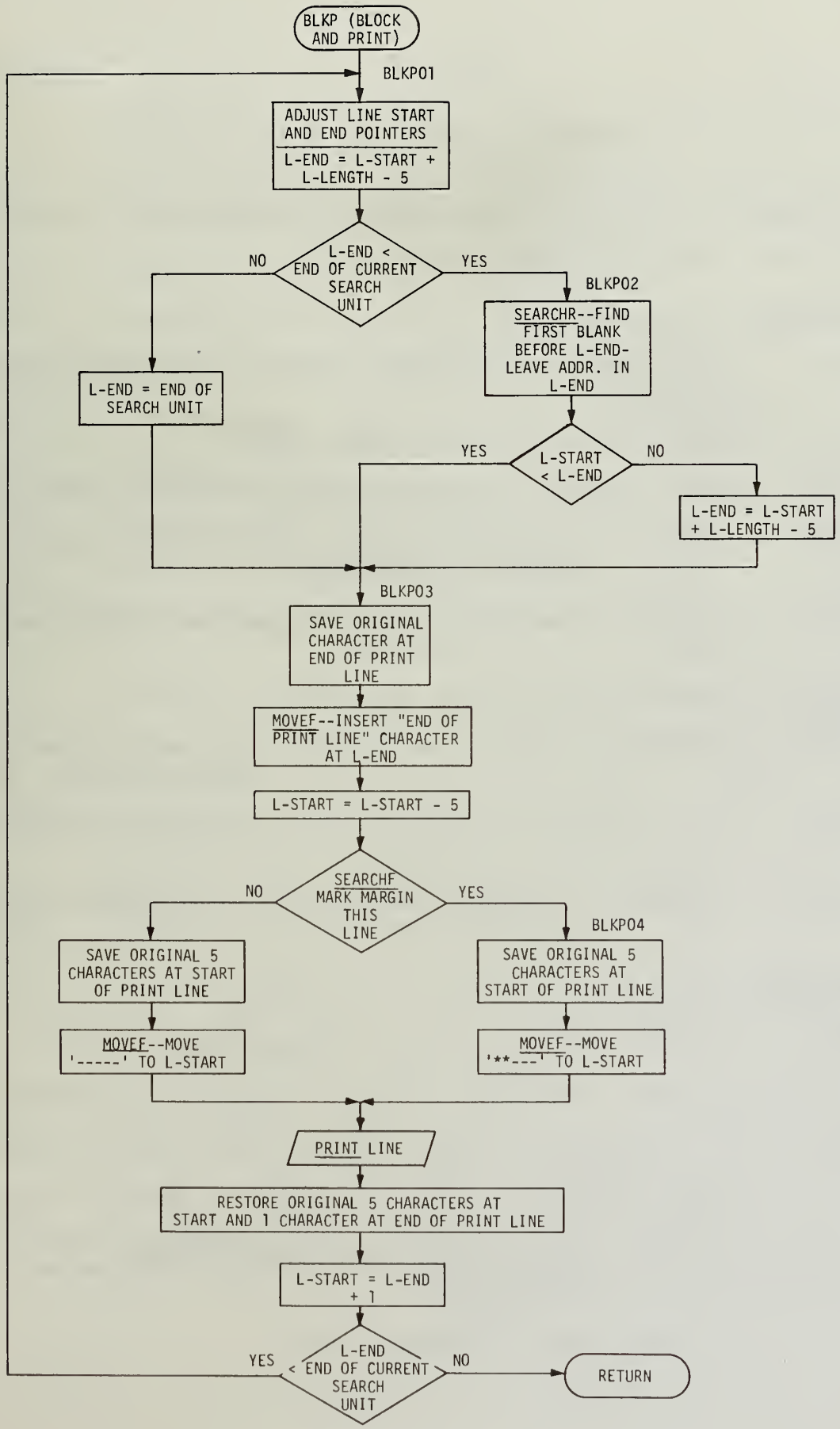


*NOTE: IF HIT IS IN ABSTRACT AND ABSTRACT HAS ALREADY BEEN PRINTED, REJECT HIT AND RESUME SEARCH AT START OF BODY.


P4
PRNT17S
CTXTEND = LEVEL 0 END
PRNT CTXT = SENT. AND FIND CTXT = PARA.
NO
YES
PRNT18A
PRNT-SRCH CTXT = PARAGRAPH
PRNT-SRCH CTXT = PRINT CTXT
SET FLAG-- FIND CONTEXT IS LARGER THAN PRINT CONTEXT
PRNT18A5
PM
RETURN BEFORE PRINTING MINOR CONTEXT
NO
YES
PRNT18B
SEARCHR--FIND START OF PRNT-SRCH CONTEXT
RETURN
SET SRCHSTRT; MOVE PRNT-SRCH CTXT END CHAR. TO SEARCH END REGISTER
PK
PRNT19
SET 'FIRST HIT', 'ORIGINAL ORDER' FLAGS
PRNT20
SEARCH FOR STRING
FLAG14
YES
SUCCESS
NO
PRNT21
SEARCHR--TO 1ST BLANK, OUTDENT CHAR. OR END OF PRINT-SRCH CTXT. CHAR. BEFORE END OF STRING
SEARCHF--FIND END OF PRNT-SRCH CTXT AND SAVE ADDRESS
MVF--INSERT 'OUTDENT' CHAR. AT BLANK
RESET 'ORIGINAL ORDER', 'FIRST HIT' FLAGS
SET 'OUTDENT' FLAG RESET 'DOCUMENT SUCCESS' FLAG
PG
CONTINUE SEARCH FOR STRING


PG
FIND CONTEXT LARGER THAN PRINT CTXT
YES
NO
OUTDENT REQUIRED
NO
YES
PRINT CONTEXT = PARAGRAPH
NO
YES
PN
(PRINT CONTEXT = SENTENCE)
PRNT22
SEARCHF--FIND OUTDENT LOCATION CHAR. OR END-OF-PARAGRAPH
PRNT23B
FIND OUTDENT MARK
NO
YES
SET SRCHSTRT
SEARCHR--TO START-OF-SENTENCE AND SAVE ADDRESS
SEARCHF--TO END-OF-SENTENCE AND SAVE ADDRESS
PRNT23
BLKP (BLOCK AND PRINT)
FIND CONTEXT LARGER THAN PRINT CTXT
NO
YES
PRINT CONTEXT = PARAGRAPH
YES
NO
PO
(PRNT24)
PN
(PRNT22)


PO
PRNT24
ARE FIND AND PRINT CONTEXTS BOTH MINOR
YES
NO
RETURN
SEARCH END CHAR. = END-OF-PRINT SEARCH CTXT
NO
YES
SEARCH END CHAR. = END-OF-BUFFER
YES
NO
HAS END-OF-PRINT-SEARCH CTXT BEEN REACHED
NO
YES
PA
(PRNT06)
HAS END-OF-BUFFER BEEN REACHED
YES
NO
PRNT25
PREPARE FOR CALL TO NBUF
PK
(PRNT19)
NBUF01I (GET NEXT BUFFER)
PD
(PRNT13)


BLKP (BLOCK AND PRINT)
BLKP01
ADJUST LINE START AND END POINTERS
L-END = L-START + L-LENGTH - 5
L-END < END OF CURRENT SEARCH UNIT
NO
YES
BLKP02
SEARCHR--FIND FIRST BLANK BEFORE L-END-LEAVE ADDR. IN L-END
L-END = END OF SEARCH UNIT
L-START < L-END
YES
NO
L-END = L-START + L-LENGTH - 5
BLKP03
SAVE ORIGINAL CHARACTER AT END OF PRINT LINE
MOVEF--INSERT "END OF PRINT LINE" CHARACTER AT L-END
L-START = L-START - 5
SEARCHF MARK MARGIN THIS LINE
NO
YES
BLKP04
SAVE ORIGINAL 5 CHARACTERS AT START OF PRINT LINE
SAVE ORIGINAL 5 CHARACTERS AT START OF PRINT LINE
MOVEF--MOVE '-----' TO L-START
MOVEF--MOVE '**---' TO L-START
PRINT LINE
RESTORE ORIGINAL 5 CHARACTERS AT START AND 1 CHARACTER AT END OF PRINT LINE
L-START = L-END + 1
L-END < END OF CURRENT SEARCH UNIT
YES
NO
RETURN

APPENDIX B

Register and Flag Assignments

A. Assignments for SUPERVISOR, FIND Decode, PRINT Decode, SHALG, and associated procedures. Unlisted registers are not used.

IAR Registers

IAR6 Return address from MARKERR; alternate return address from LWTAG and LONCHN

IAR7 Return address from SORT, LWTAG, LONCHN

IAR8 Return addresses from major routines (FIND, SHALG, SRCH, etc.)

IAR10 =SUP50-1: Transfer address for printing "Invalid character or Keyword" error message

IAR11 "FAILURE" return address for TSRCH (searches in RESKEY and CTXT Tables)

IAR12 "SUCCESS" return address for TSRCH

IAR13 =PRINT02-1, =PRINT05-1 or =FIND03-1: New character processing in decode routines

IAR14 Error message address for "missing (')" exit from SEARCHF instruction in LTSTRING processing

IAR15 =LTSTR02-1: successful comparison exit from COMPARE instruction in LTSTRING processing

PTR Registers

PTR0 In FIND: Temporary "new" character address in LEGALTAB

In SHALG: Tag Table address of first entity with current tag

PTR1 In FIND and PRINT Decode: Byte address of current input character

In SDMP, SCHNG, and PACK: Word address of next word to be processed

In SHALG: "Chain locater" -- Normally points to second column of Tag Table entry for parenthesized expression at start of current chain

In LWTAG, LONCHN: Temporary pointer

PTR Registers

PTR2 Points to Hex :81: in CHR 14. (Used to move Hex :81: ahead of alphanumeric keystrings for table searching)

PTR3 Points to Hex :82: in CHR13. (Used in TSRCH to check for :82: at end of data string)

PTR4 Points to Hex :20: (Blank) in CHR11

PTR5 In MARKERR: Data pointer for error message preparation

In SHALG: Utility pointer into Tag Table

PTR6 In PRINT Decode: Local functions in initial sections

In SHALG: Utility; special stack pointer

In SHALG, LWTAG, LONCHN: Pointer to current tag

PTR7 In SHALG: Utility pointer in Tag Table and Stack

PTR8 In PRINT Decode: CTXT Table pointer for "Clear Context Table" procedure

In SHALG: Next location to receive a link in constructing re-ordered chain and in SUCCESS/FAILURE assignments

PTR9 Points to Hex :80:, the End-of-Table symbol, in CHR15

PTR10 Points to Hex :27: (Apostrophe) in CHR2

In SHALG: Used in stacking individual entities

PTR11 In SHALG: Utility; special stack pointer

PTR12 In FIND: Standard Data Pointer for string instructions (Key Pointer in "skip blanks" instructions)

In SHALG: Temporary stack pointer

PTR13 Start of character table, "CHARS"

PTR14 Current legal character address in LEGALTAB

PTR15 In TSRCH: Temporary storage for contents of PTR1

In SDMP, SCHNG and PACK: Points to DUMP instruction under construction or to S-Memory words being changed

CHR Registers

CHR0	"FIND" context start character
CHR1	"FIND" context end character
CHR2	In FIND Decode: Hex :0027: (Apostrophe)
	In PRINT Decode: Hex :002C: (Comma)
CHR3	Hex :005E: (↑ for error messages in MARKERR)
CHR11	Hex :0020: (Blank)
CHR12	Hex :000D: (Carriage Return)
CHR13	Hex :0082: Alphanumeric string suffix in keyword-type tables
CHR14	Hex :0081: Alphanumeric string prefix in keyword-type tables (including teletype input line, TTYIN)
CHR15	Hex :0080: End-of-table symbol

CTR Registers

CTR0	Character identification in PRINT Decode and FIND Decode
CTR1	In FIND: Total character count for requested search term
	In SHALG: Counter--total number of different tages on current chain
CTR2	In MARKERR: Character count for error message preparation
	In SHALG: Counter--number of parenthesized expressions with current tag
CTR3	In SHALG: Counter--number of tags on current chain which represent strings only
CTR4	In SHALG: Counter--number of strings associated with current tag
CTR5	In SHALG: Counter--number of strings inside a "new" parenthesized expression
CTR6	In SHALG: Temporary storage to assist in setting up stack pointers; total number of entities on current chain
CTR7	Hex :008D: (Used in CTXT Table to identify contexts selected for printing)
CTR9	In SDMP, SCHNG, and PACK: Counter for number of passes through PACK procedure

CTR Registers

CTR9 In SHALG: total number of entities on current chain--used as counter in constructing "sorted" chain in Tag Table

CTR15 Hex :0000:

FLAGS (Low order bit is numbered 15)

BIT15 Search Success: On return from search, 1 implies NOT FOUND, 0 implies at least one responding document

In SHALG: indicator for "local" conditions

B. Assignments for SRCH, PRNT, and associated procedures.

IAR Registers

IAR0 In SRCH: Used as return address set by DECRV instruction

IAR1 =SRCH80-1: Transfer address for "CONTINUE?" message

IAR3 Return address from BLKP (Block & Print)

IAR4 Return address from NEXTM

IAR5 Return address from NEXTL

IAR6 Return address from HUNT, NBUF

IAR7 Return address from PRNT

IAR8 Return address to SUPERVISOR

IAR9 =SRCH32-1: "FIND Level-0 end character" exit from SEARCHF instruction at SRCH31A

IAR11 "FAIL" transfer address for FIND instruction at HUNT03A

IAR13 =PRINT23B: "No more outdent marks" exit for SEARCHF instruction at PRNT22

IAR14 =SRCH90-1: transfer address for "SYSERR" message

IAR15 In PRNT: Temporary storage for PTR2 during some calls to BLKP

PTR Registers

PTR0	In SRCH: "ADENDL1" (Address of the End of the Level 1 search context)
PTR1	In BLKP: Start of print buffer
	In SRCH: Text of string currently sought; temporary storage
	In NEXTL, NEXTM: Return argument--PTR1= pointer to next string; if none, then PTR1=0
PTR2	In BLKP: End of print context
	In SRCH: Search start pointer (Different from search start variable, SRCHSTRT, in PTR6)
PTR3	In SRCH, HUNT, NEXTL, NEXTM: Current Tag Table entry
PTR4	In BLKP and PRNT: Temporary storage (the two uses are independent of one another)
PTR5	In PRNT: Points to <<CTR5>> (Hex :84:, outdent symbol)
PTR6	SRCHSTRT
PTR7	In PRNT and BLKP: Points to <<CTR4>> (Hex :89:, end of print buffer symbol
PTR8	In PRNT: Print context table pointer
PTR9	Points to :80: (End-of-table symbol) in CHR15
PTR11	In HUNT: Temporary storage for PTR3 during calls to NEXTM
	In PRNT: Temporary; used to transfer context delimiters from CTXT Table to CHR registers; search pointer for "Insert 'MARK LINE' Symbol"
PTR13	In BLKP: <SAVEBLOK> (used to save and restore characters displaced by (**---) or (-----) or (:89:) in print buffer
PTR14	In PRNT: Points to End-of-print context, if in core (large value, otherwise)
PTR15	In BLKP: Points to <LINEMARK> or <LINEMARK+1>

CHR Registers

CHR0	"FIND" context start character
CHR1	"FIND" context end character
CHR2	SEARCH Level-0 start character
CHR3	SEARCH Level-0 end character
CHR4	SEARCH Level-1 start character
CHR5	SEARCH Level-1 end character
CHR6	SEARCH Level-2 start character
CHR7	SEARCH Level-2 end character
CHR8	(current) PRINT context start character
CHR9	(current) PRINT context end character
CHR10	END-of-SEARCH character
CHR11	PRINT-SEARCH context start character
CHR12	PRINT-SEARCH context end character
CHR13	Hex :0020: (Blank)
CHR14	Hex :0088: (End of buffer symbol)
CHR15	Hex :0080: (End of table symbol; end of STRING in STRTXT Table)

CTR Registers

CTR0	Used by every string instruction
	IN PRNT: number of characters for printed line
CTR1	Number of disk sectors currently in core (or number about to be read)
CTR2	Number of disk sectors remaining to process
CTR3	In NBUF: Character at start of required text
CTR4	Hex :0089: (End of print buffer)
CTR5	Hex :0084: (Outdent symbol)
CTR7	Hex :008D: (Used by PRNT in searching CTXT Table)
CTR8	In PRNT: CTXTEND (End character for current major print CTXT or last major CTXT printed)
CTR10	Print control flags (Copy of Flags 8-12: Positions 10-12 may be changed by PRNT as context changes)
CTR11	Reserved for search control flags (Copy of Flags 8-12)
CTR12	Hit count (Used in conjunction with "CONTINUE?" message)
CTR13	125 (Maximum number of text characters/printed line)
CTR14	Hex :0004:
CTR15	Hex :0000:

FLAGS (Low-order bit is numbered 15)

BIT0	First Entry (to PRNT)
BIT1	Last Entry (to PRNT)
BIT2	Buffer changed (Used in PRNT)
BIT3	Special condition indicator: This bit = 1 if and only if "PRINT" context is minor (sentence or paragraph) and "FIND" context is major or minor, but larger than "PRINT" context
BIT4	Outdent Flag
BIT6	If this bit = 1, return from first call to PRNT before printing individual sentences or paragraphs
BIT7	First Hit (Return from HUNT after first success in locating any requested string)
BIT8*	"FIND" is a major context section
BIT9*	"FIND" is SENTENCE
BIT10*	{ALL / CURRENT} "PRINT" context is a major context section
BIT11*	{SOME / CURRENT} "PRINT" context is SENTENCE
BIT12*	"FIND" context is a Bibliographic Data subsection
BIT13	Search Mode: 0 implies "Revised order" - 1 implies "Original order"
BIT14	Document Success: If this bit = 1, the document currently being searched satisfies the search request
BIT15	Search Success: If this bit = 0 after a search, some document in the data base satisfies the search request

*Flag bits 8-12 appear in Counter Register 11 instead of the variable FLAGS.

LIST OF REFERENCES

[1] Hirohide Yamada, "Emulation of Disk File Processor", University of Illinois at Urbana-Champaign, Department of Computer Science Report No. 436, June 1971.

[2] E. J. Polley, Jr., "An Assembler for Efficient File Manipulation", University of Illinois at Urbana-Champaign, Department of Computer Science Report No. 534, August 1972.

[3] W. E. Caves and R. E. Tomlinson, "The Decision Module Compiler", Management Information Services, Detroit, Michigan, 1970(?).

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-74-657	2.	3. Recipient's Accession No.

4. Title and Subtitle

AN EXPERIMENTAL INFORMATION RETRIEVAL SYSTEM

5. Report Date: July 1974

6.

7. Author(s): William Howard Stellhorn

8. Performing Organization Rept. No. UIUCDCS-R-74-657

9. Performing Organization Name and Address

University of Illinois at Urbana-Champaign
Department of Computer Science
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.: US NSF GJ-36936

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, D. C.

13. Type of Report & Period Covered: Technical Report--1974

14.

15. Supplementary Notes

16. Abstracts

An experimental retrieval system designed to support data bases with little inherent structure is described. The initial data base contains the full text of several technical articles on information retrieval. Searching proceeds by means of a direct sequential scan through the data, and the user has very general control over the structure and context of the search. The system runs on a microprogrammable minicomputer, and several text searching and manipulation commands have been implemented in microcode.

Part I of the report is a user's guide. Part II contains detailed technical descriptions of algorithms employed for parsing, scheduling and executing a search request.

The system will be used to analyze strategies for efficient searching in this environment and to study the interaction between the system and a group of motivated users under controlled conditions.

17. Key Words and Document Analysis. 17a. Descriptors

Information Retrieval
Interactive Systems
Full Text Scanning
Algorithms
Microprogramming
Scheduling
Parsing

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
RELEASE UNLIMITED	UNCLASSIFIED	119
	20. Security Class (This Page) UNCLASSIFIED	22. Price